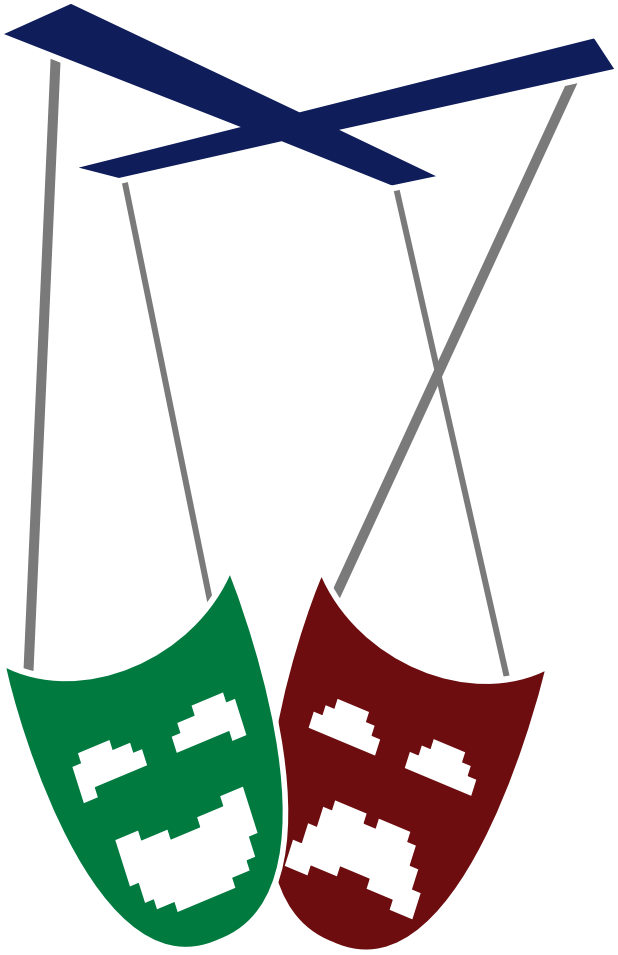
Realization and high level specification of facial expressions for embodied agents

Ronald Paul

Master's Thesis
Human Media Interaction
Faculty EEMCS
University of Twente

Graduation Committee
Job Zwiers
Dennis Reidsma
Herwin van Welbergen

Enschede, June 2010



UNIVERSITY OF TWENTE.

Abstract

In this thesis we describe work done related to realization and high level specification of facial expressions for embodied agents. Realization is done by implementation of MPEG-4 Facial Animation. High level specification of facial expressions is done by creating FACS (Facial Action Coding Standard) configurations or by choosing points on a circular emotion space.

For realization of facial expressions, an editor has been developed which can be used to set face dependent parameters like feature point location and other variables that control the way a face is deformed. Evaluation of our implementation of MPEG-4 Facial Animation is done by comparing it to several other virtual faces that have implemented it. This shows that our implementation is performing better than any of the faces it was compared to.

We visually show that expressions created using the FACS high level specification method are corresponding to real life imagery very well.

Preface

This document is written in the context of the final graduation project of my Master's degree program Human Media Interaction. The thesis is now complete and I learned a lot during the process. This final project took me a few months more than it takes the average student to complete but I will conveniently not mention that again. It is a milestone. It marks the end of almost a decade of tertiary education and the beginning of something completely new.

Enough sentimentalities. I would like to thank a few people that enabled or helped me with my final project. First of all, friends and family in general. Not only for asking me about my progress very regularly of course. I also thank my parents for their continuous support through the years.

I specifically would like to thank the graduation committee. Herwin van Welbergen for his help with BML, Dennis Reidsma for his help in software development and deployment and Job Zwiers for helping me from the very beginning when my wishes were vague and needed to be concertized in a real graduation project. And all these three persons for guiding and assisting me in the project and giving comments and tips for improvement of the preliminary versions of this thesis.

Ronald Paul
Enschede, June 2010

Contents

Chapter 1

Introduction

Computers have been around for a while. The interface with human users of these computer systems is a integral part of the system for many applications. Most of these interfaces were mostly task oriented in the past, but user centric approaches became possible with development of more powerful two and three dimensional graphics capabilities. A computer will always be a computer but work with computers becomes more pleasant and efficient if a human user is confronted with a visually appealing virtual human.

An embodied conversational agent (ECA) can be constructed to show emotions and affect to improve user experience. This means that facial expressions should be displayed on the ECA's face. Facial expressions have different representations and there is a trade-off between the number of control parameters of a particular representation and the graininess of control. A high number of parameters allows more subtle expressions but is more time consuming. We chose for a basic expression representation that actually specifies how the face should be altered and two higher level representations that can be translated in this basic one.

1.1 Background

The research performed at Human Media Interaction (HMI) is focused on interaction between humans and machines. With the use of ECAs, the user faces a whole new kind of interface compared to the mature standard graphical user interfaces. The user actually is able to get acquainted with and develop affect for a character that acts such as a real human which in it self poses a whole range of advantages in the field of human-computer interaction such as more pleasure and less stress.

The face is the most important part of the human body for communication with other humans. Not only for verbal but also for non-verbal communication such as expressions.

Non-verbal communication for virtual characters require a vast amount of facilities. From the mental model of emotion, how these emotional states changes over time and how it is influenced by stimuli from other characters or the user to the actual visualization of expression by changing the virtual world face by adjustments. This research is focused on the facilities that come last in line: the representation of expressions and the translation in adjustments to the virtual face.

When the virtual face is able to show expressions, animation is the next step in the process of building usable ECAs. The only link this research has with this future work is the work done for reading BML, a markup language for describing human behavior.

1.2 Objectives

The objectives for this project are to perform research in the context of and design and implement a set of tools that:

1. assist in creating facial expressions by providing several high level steering instruments that can be driven by a limited number of parameters;
2. provide a good trade-off between number of control parameters and range of expression;
3. actually apply adjustments to virtual faces in such a way that these faces are interchangeable with other faces without changing too much of the expression;
4. interface with Behavior Markup Language (BML) and
5. work in real-time.

1.3 Approach

The work started with a literature research. This brought up earlier used methods and techniques for implementation of facial animation. Putting them side to side enabled us to make a decision about what facial expression representations to use, what translations between them are possible and how to apply adjustments to the virtual face.

Pieces of software are developed to facilitate reading of BML, translation of high level expression representation into lower level ones, and application of adjustments to virtual faces. Some of these pieces are integrated so they can work together.

To show the correctness of implementation of the chosen method for applying adjustments, produced faces are placed side to side to other faces that have implemented the same method and scored for all possible steering parameters.

To show the effectiveness of the expression representation translations, the aforementioned implementation are used to show outputs for a selection of the possible higher level steering parameters.

The literature research brought up a part of the MPEG-4 standard, named Facial Animation (FA) as low level representation of expression and application method. For two higher level representations, one called Facial Action Coding Standard (FACS) and the other being an emotion model by Plutchik [21], translations into MPEG-4 FA have been used.

1.4 Structure of the report

Chapter 2 starts of with description of literature found relevant for this project. It describes the different techniques for animation of faces, pseudo muscle-based animation and different conversions or translation between expression representations.

BML is described in chapter 3, along with a description of the design of software that facilitates reading it. Also, a part about scheduling of behaviors is included.

MPEG-4 FA is described in chapter 4. It includes a description of Xface that is MPEG-4 FA compatible and the design and evaluation of the realized prototype for for application of micro adjustments to the face and setting parameters that depend on the virtual face.

Chapters 5 and 6 describe two conversion prototypes, from FACS to MPEG-4 FA and from emotion to MPEG-4 FA. Both chapters describe the procedure of conversion, discuss the design of the software prototype and evaluate the performance of the conversions themselves.

In chapter 7, some of the weaknesses and strengths of a few of the methods that have been developed or reused, are discussed.

Chapter 8 concludes this thesis and describe possible future research.

Chapter 2

Literature

Some important choices are made based on the information found in literature. This chapter reviews these sources and gives an overview of animation techniques, pseudo muscle-based animation and conversions from high level expression representations into lower level ones.

2.1 Animation techniques

A face is represented by a 3D mesh with a varying number of vertices. Ultimately, facial animation is about moving the vertices in space over time. The problem lies in the fact that since all vertices have three degrees of freedom and even low resolution meshes already have hundreds of vertices, the total amount of possible combinations of movements of the whole face is very large. Even when just looking at a face in the real world, it is hard to mimic these movements.

This is why, during the long history of facial animation, a lot of different techniques have been developed for animating a face. The remainder of this section has been used to describe fundamental approaches, to give some examples of techniques and to work out the direction into which the technique for this thesis should develop.

There is more then one categorization possible. Parke et al. uses these categories: interpolation, performance-driven, direct parameterization, pseudo muscle-based and muscle-based animation [19]. Although this work has proven to be a good starting point for research occurred later in time, I find the taxonomy developed by Ersotelos and Dong [10] more intuitive. This survey of realistic facial modeling and animation approaches the facial animation with only three categories: blend shape based animation, performance driven animation and simulation. I added a fourth low-level category for two almost ancient techniques.

2.1.1 Low level animation

One of the first and probably the oldest way of animating a face was to manually pick vertices, give them other positions and gradually apply this displacements over time. This was a lot of work, even for the first facial surfaces with a low number of polygons. And since the average number of vertices in a face have increased a few orders of magnitude since then, this method has became infeasible.

Another approach within this category is direct parameterization. It is still based on interpolation and key-frames but a face can be controlled by a much smaller set of parameters. Every parameter has a specific influence on the face, a numeric range and can be interpolated over time. One of the first attempts of direct parameterization by Parke is described in [12]. The challenge is to determine a good set of parameters and to implement them correctly.

The advantage of direct parameterization is that once control parameters are determined, they provide a detailed control over the face. But determining this is hard. Complexity of creating an animation with these control parameters is related to the number of control parameters, as is the possible range of expressions.

2.1.2 Blend shape based animation

Blend shape based animation is a simple technique for animating a face. Multiple meshes (key poses or blend shapes) are created, for example a neutral one and one for each of the basic emotions anger, disgust, fear, joy, sadness and surprise proposed by Ekman et al.[9]. All meshes contain vertex positions for the same vertices. When a face should show anger, the position of all vertices can easily be interpolated between the positions of the neutral face and the angry face. With respect of course to the preferred duration of the total animation and the time already elapsed since the beginning of the animation.

It is also possible to create key poses with subsets of the vertices available. Consider one pose for a smile and one pose for raised eyebrows. This way, multiple poses can be combined and blended to get a somewhat more flexible face.

This technique has one obvious advantage: it is simple. Once the meshes are available, it is trivial to create software that incorporates facial animation by interpolation. As a consequence, it is also computationally cheap. However, to obtain more fine-grained animation, lots of key poses are needed and creating them is labor intensive. Furthermore, it is not possible to create expressions that are outside the bounds of the set of created key poses. Extrapolation can help in this case but is dangerous.

2.1.3 Performance-driven animation

When features from a real human face are extracted and used for animation, we call this performance-driven animation. They often use specialized input devices such as a laser scanner or use video based motion tracking.

Performance-driven animation can result in a realistic facial animation. But it is hard to create a system that handles all data from input devices correctly. Furthermore, data is difficult to use in a generic way. Once recorded material is to be used on different (virtual) faces, data is abstracted and loses its fine detail that partly enabled the reality. Furthermore, it requires extra hardware and a real life actor.

2.1.4 Simulation

Simulation techniques recreate or approach the workings of one or more anatomical structures.

The technique of muscle-based animation is a simulation technique and tries to mimic important anatomical structures of the head such as bone, tissue, muscles and skin. This approach should give us a limited set of control parameters that, although they are low in number, provide a good range of expressions.

Waters introduced a muscle model [3, 12] that describes two types of muscles: linear or parallel muscles that pull and sphincter muscles that squeeze. Muscles of the first type have a attachment point and a zone of influence. Per node, displacement is calculated from the distance from the attachment points, the properties of the zone of influence, elasticity of the 'skin' and the angle with the center of the zone of influence.

Complete muscle based animation is a technique that should give realistic results if and only if the anatomical human face structures are recreated with enough detail. However, computationally it is prone to be too complex to perform in a real-time environment. We will not go into this any further.

Pseudo muscle-based animation on the other hand only models muscles. This is a kind of direct parameterization. Further simplification can be done by omitting small muscles with diminishable influence or by using an abstraction of all possible movements of the face. Two of those abstractions are FACS and MPEG-4 FA. The advantage of pseudo muscle-based animation is that it provides a good ratio between control parameters and range of possible expressions. §2.2 goes deeper into FACS and MPEG-4 FA.

When creating an animation of a face, temporal information can help in recognition of the expressions. When a face is smiling, it takes time before the smile is fully realized and all muscle contractions follow specific curves. Trapezoid functions are widely used. These have three linear stages: *application*, *release* and *relaxation*. It is shown that the

actual shape is more complex [11], but trapezoid functions are still popular because there is insufficient evidence for what these more natural movements actually are [23]. We did not perform any further research on this topic of incorporation of temporal information because our goal was not to create animations but only static expressions.

2.2 Pseudo muscle-based animation

2.2.1 FACS

FACS stands for Facial Action Coding System. It was developed by Ekman and its colleagues [8] and consists of a number of Action Units (AUs). The goal was to create a comprehensive system in which all visually distinguishable facial movements are described. Although it has its origin in psychology, it has been adopted by facial animation synthesis systems.

FACS was created by determining which of the facial muscles can be used voluntarily and independently and to determine how much a muscle changes facial appearance. There is a many to many relation between AUs and facial muscles. See Table 2.1 for some of the AUs with most of them referencing one or more specific facial muscles. A full reference can be found in Appendix A. Most of the muscles can be visually identified with Figure 2.1.

All AUs can be used at any time with only a few restrictions: some AUs conflict with each other (they work in the opposite direction) and some AUs hide the visual presence of others.

AU	Description	Facial muscle
1	Inner Brow Raiser	Frontalis, pars medialis
2	Outer Brow Raiser	Frontalis, pars lateralis
4	Brow Lowerer	Corrugator supercilii, Depressor supercilii
5	Upper Lid Raiser	

Table 2.1: A few Action Units defined in FACS. A full reference can be found in Appendix A.

2.2.2 MPEG-4 Facial Animation

A relatively recent standard, as opposed to FACS, for Facial Animation, is MPEG-4 FA. It defines Feature Points (FPs), Facial Action Parameters (FAPs) and Facial Action Parameter Units (FAPUs). All of these terms are explained in 4.1. There are others, such as Facial Description Parameters (FDPs), Face Interpolation Tables (FITs) and Face Animation Tables (FATs), but those are only relevant in cases where facial animation is embedded in a MPEG-4 stream just as in streaming video.

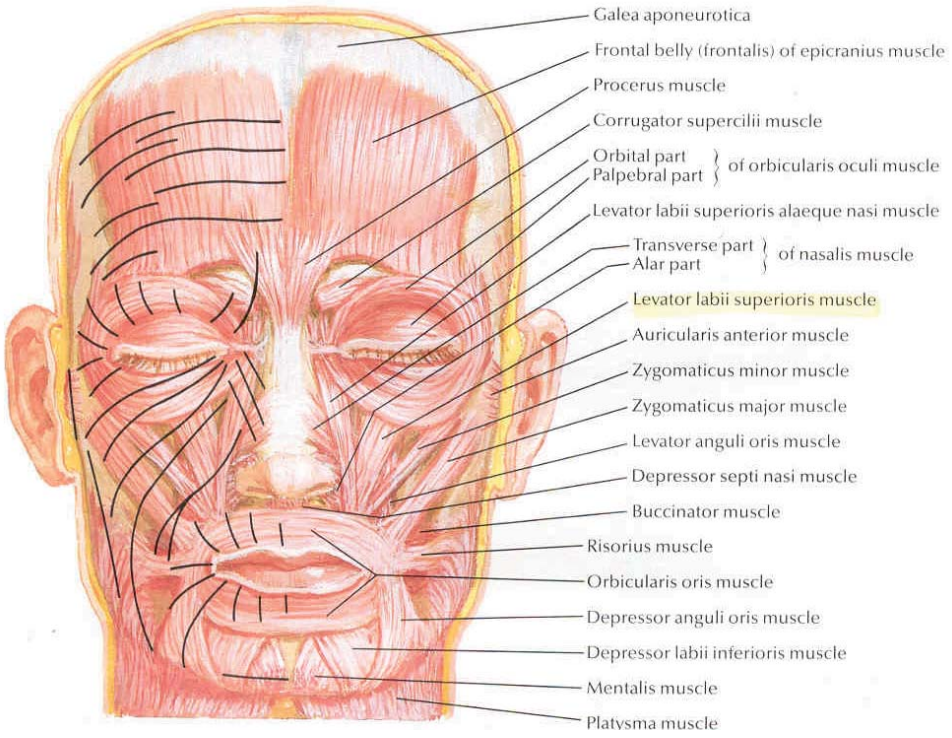


Figure 2.1: Facial muscles.

MPEG-4 has at least one limitation; it is not possible to reposition the points at the lower base of the nose directly. This portion of the face is a key-indicator for the disgust-emotion. Disgust can still be made visible through MPEG-4 FA though.

Possibilities for higher level control

Most of the FAPs are low level control points. They only control a small region of the face. The first two FAPs however are more high level. FAP 1 allows to apply an expression (anger, disgust, fear, joy, sadness and surprise) and FAP 2 allows to apply a viseme (or a set of visemes). Also, Raozaiou, Tsapatsoulis, Karpouzis and Kollias [22] propose a method for creating intermediate facial expressions. See §2.3.4 and §6.

Comparison with FACS

MPEG-4 FAPs are strongly related to FACS [22]. Creating archetypal expressions in FAPs traditionally has been performed by analyzing which FACS AU are fired [16].

MPEG-4 facilitates animation independent of face models because it makes use of FAPUs. How muscle tensions for FACS correspond to specific offsets of portions of the face is undefined and therefore it is hard to create a animation that does not depend on the implementation of FACS let alone the face (dimensions, topography, etc.) it is applied to.

Implementation

Although all feature points (see Figure 4.2) are clearly defined, application of those points to a actual 3D mesh of a face and moving them is not trivial. Choosing vertices that correspond to feature points should be easy with the points around the mouth and eyes. Points such as 5.4 or 5.2 can be estimated since their influence on surrounding vertices does not depend on an exact placement as the points in the corners of the eye or mouth do.

As of moving a point, there is more to it than just moving the vertex. There must be a mechanism that move the surrounding points in a natural way. According to the MPEG-4 FA book [18], the "mapping of feature points motion onto vertex motion can be done using lookup tables such as FAT, muscle-based deformation, distance transforms or cloning from existing models". For the rest of this section, we will shortly look into some of these methods.

Face Animation Tables (FATs) define how vertices of a model are displaced as a function of the FAP. With them, displacements for each individual vertex that surround a feature point for the whole range of the FAP can be defined.

Bee et al.[6] use a fully controllable virtual head which was developed by Augsburg University. This head (Alfred) had predefined morph targets for all FACS action units. Although FACS is used, a similar approach can be used for MPEG-4. This can yield realistic results but only if the morph targets themselves are realistic.

Xface is, amongst other things, an open source implementation of MPEG-4 FA [5]. In Xface, users should select a set of vertices for all feature points (zones). When moving a feature point, it uses a raised cosine function to deform the zone and displace the vertices in the zone. This is a distance transform and should achieve satisfactory results [5]. Kojekine et al. [14] use Compactly Supported Radial Basis Functions (CSRBF) as a mean for 3D deformation. Free form deformation (FFD) could also be used, see Kalra et al. [13].

Another real-world example of implementation of MPEG-4 FA is Greta. In addition, it features an ad-hoc technique for creating wrinkles [20]. See Figure 4.18 for a screenshot. It displaces vertices based on the distance to the feature point using a sinoidal function.

2.3 Conversions

2.3.1 Introduction

This section describes the higher level control mechanisms for synthesis of facial animation based on simulation. We have already established that FACS and MPEG-4 both are representations for expressions on the face. But for an animator, manually controlling AUs or FAPs is still too much work.

In this thesis, *emotion* is defined as a state of mind which results in one or more *expressions*.

2.3.2 From emotion to FACS

Zhang, Ji, Zhu and Yi [27] made a simple mapping from each of the six basic emotions to Action Units that are active for this emotion. See Figure 2.2. This mapping is not quantitative. For example, when we want to make a sad face, we know that we at least need AUs 1, 15 and 17 but we don't know what intensities are appropriate.

Expressions	Primary AUs	Auxiliary AUs
Happiness	6, 12	25, 26, 16
Sadness	1, 15, 17	4, 7, 25, 26
Disgust	9, 10	17, 25, 26
Surprise	5, 26, 27, 1+2	
Anger	2, 4, 7, 23, 24	17, 25, 26, 16
Fear	20, 1+5, 5+7	4, 5, 7, 25, 26

Figure 2.2: Activated AUs for each of the six basic emotions. Taken from Zhang et al. [27].

2.3.3 From FACS to MPEG-4 FA

AUs and FAPs are strongly related [27, 22]. Zhang et al. [27] related all relevant AUs to FAPs. See Figure 2.3. It should be relatively easy to construct a complete map for all AUs.

We actually implemented this conversion, see §5 for a detailed description on the approach taken.

2.3.4 From emotion to MPEG-4 FA

Raouzaiou, Tsapatsoulis, Karpouzis and Kollias [22] describe a method for enriching human computer interaction, focusing on analysis and synthesis of primary and intermediate

FAP Number	FAP Name	Distance of Two Feature Points	FAPU	AU
31	raise_l_i_eyebrow	$D_y(4.2, 3.8)$ [3]	ENS	AU1
32	raise_r_i_eyebrow	$D_y(4.1, 3.11)$	ENS	
35	raise_l_o_eyebrow	$D_y(4.6, 3.12)$	ENS	AU2
36	raise_r_o_eyebrow	$D_y(4.5, 3.7)$	ENS	
31_	raise_l_i_eyebrow [1]	$D_y(4.2, 3.8)$	ENS	AU4
32_	raise_r_i_eyebrow	$D_y(4.1, 3.11)$	ENS	
37	squeeze_l_eyebrow	$D_x(4.4, 3.8)$	ES	
38	squeeze_r_eyebrow	$D_x(4.3, 3.11)$	ES	
19_	open_t_l_eyelid	$D_y(3.6, 3.2)$	IRSD	AU5
20_	open_t_r_eyelid	$D_y(3.5, 3.1)$	IRSD	
19	close_t_l_eyelid	$D_y(3.6, 3.2)$	IRSD	AU6
20	close_t_r_eyelid	$D_y(3.5, 3.1)$	IRSD	
41	lift_l_cheek [2]	$D_y(5.4, 3.12)$	ENS	
42	lift_r_cheek	$D_y(5.3, 3.11)$	ENS	
21	close_b_l_eyelid	$D_y(3.4, 3.6)$	IRSD	AU7
22	close_b_r_eyelid	$D_y(3.3, 3.5)$	IRSD	
61	stretch_l_nose	$D_y(9.14, 3.8)$	ENS	AU9
62	stretch_r_nose	$D_y(9.13, 3.11)$	ENS	
59	raise_l_cornerlip_o or	$D_y(8.4, 3.12)$	MNS	AU10
60	raise_r_cornerlip_o	$D_y(8.3, 3.11)$	MNS	
59	raise_l_cornerlip_o	$D_y(8.4, 3.12)$	MNS	AU12
60	raise_r_cornerlip_o	$D_y(8.4, 3.11)$	MNS	
53	stretch_l_cornerlip_o	$D_x(8.4, 9.15)$	MW	
54	stretch_r_cornerlip_o	$D_x(8.3, 9.15)$	MW	
59_	lower_l_cornerlip	$D_y(8.4, 9.15)$	MNS	AU15
60_	lower_r_cornerlip	$D_y(8.3, 9.15)$	MNS	
5	raise_b_midlip	$D_y(8.2, 9.15)$	MNS	AU16
16	push_b_lip	$D_y(8.2, 8.1)$	MNS	
18	depress_chin	$D_y(8.2, 9.15)$	MNS	AU17
53	stretch_l_cornerlip	$D_x(8.4, 8.3)$	MW	AU20
54	stretch_r_cornerlip	$D_x(8.3, 8.4)$	MW	
5	raise_b_midlip	$D_y(8.2, 9.15)$	MNS	
53_	tight_l_cornerlip	$D_x(8.4, 8.3)$	MW	AU23
54_	tight_r_cornerlip	$D_x(8.3, 8.4)$	MW	
4	lower_t_midlip	$D_y(8.1, 9.15)$	MNS	AU24
16	push_b_lip	$D_y(8.2, 9.15)$	MNS	
17	push_t_lip	$D_y(8.1, 9.15)$	MNS	
3	open_jaw (slight)	$D_y(8.2, 8.1)$	MNS	AU25
5_	lower_b_midlip (slight)	$D_y(8.2, 9.15)$	MNS	
3	open_jaw (middle)	$D_y(8.2, 8.1)$	MNS	AU26
5_	lower_b_midlip (middle)	$D_y(8.2, 9.15)$	MNS	
3	open_jaw (large)	$D_y(8.2, 8.1)$	MNS	AU27
5_	lower_b_midlip (large)	$D_y(8.2, 9.15)$	MNS	

Figure 2.3: Mapping between FAPs and AUs. Taken from Zhang et al. [27].

facial expressions. An important asset for this method is the emotion wheel by Plutchik [21], see Figure 2.4.

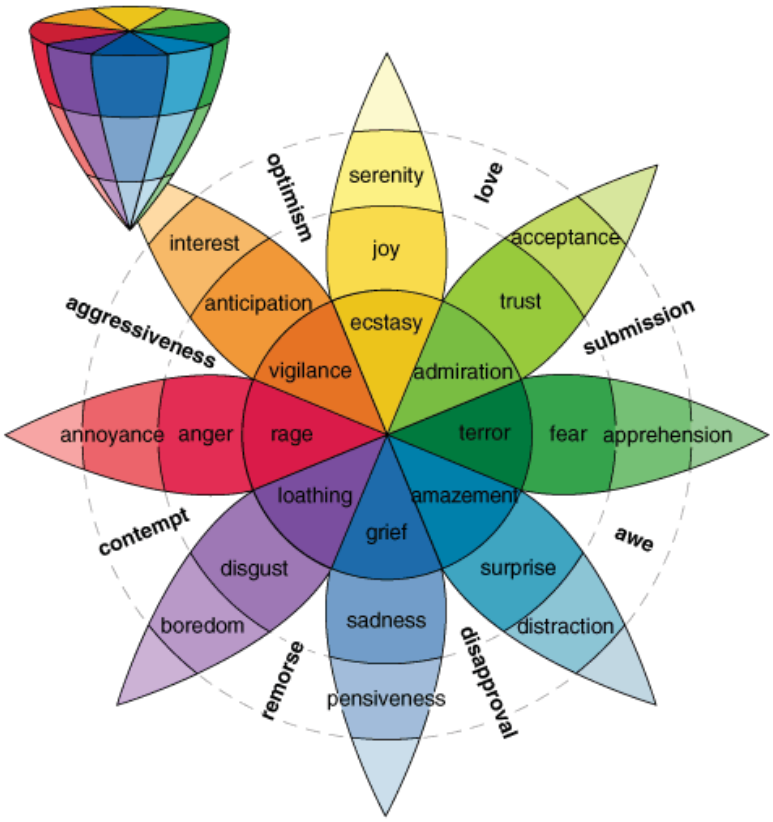


Figure 2.4: Plutchik's model of emotion. It describes relations among emotion concepts.

Raouzaiou et al. have build profiles each belonging to a certain archetypal expression. All archetypal expressions have a coordinate on Plutchik's model of emotion, and the method supplies us with a procedure to calculate a FAP-configuration for the complete coordinate space. The complete procedure and background information on this emotion space can be found in chapter 6.

Chapter 3

Behavior Markup Language

This chapter describes Behavior Markup Language (BML), the design of a parser that reads BML and stores it in an internal representation and includes some words on scheduling of behaviors. It is part of SAIBA [15], which is short for Situation, Agent, Intention, Behavior and Animation. The goal of its creators is to have a uniform framework for multimodal generation. This should reduce the overall time researchers spend creating their own languages, interfaces and architectures and encourage cooperation because modules now can be shared easily.

On first sight, BML might look like it has nothing to do with animating a virtual face, but with the facilities it contains, facial expression can be specified and - over time - facial movement. In this project, it is the primary way to drive the facial animations. And because BML is not fully specified yet, this project can help in maturing it when it comes to facial animation. For general use withing HMI, a BML recursive descent XML parser was designed and implemented.

3.1 BML

BML is part of the SAIBA framework. The structure of the framework is depicted in Figure 3.1.

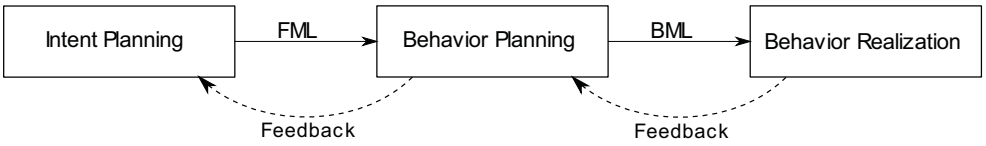


Figure 3.1: Overview of the SAIBA framework.

The framework divides multi-modal generation over three levels:

1. Intent planning
2. Behavior planning
3. Behavior realization

Two major interfaces between these levels are:

1. Function Markup Language (FML)
2. Behavior Markup Language (BML)

We will only describe BML since Behaviour Planning uses facial expression and animation for the first time in the whole SAIBA process.

BML is XML. The top level element is `<bml>`. The standard [1] defines the core. Researchers are free to create their own extensions in special tags or in separate namespaces. These additions are called beyond core. In this container, one or more of the following core tags may be placed:

- `<head>`: movement of the head. Supports nodding, shaking, tilting and rhythmic movement.
- `<gaze>`: angular movement of the eyes, so that can be controlled where a character is looking at.
- `<locomotion>`: used to move the body of an character from one location to another.
- `<posture>`: used to put the body of an character into a specific posture (standing, sitting, lying, etc.)
- `<speech>`: specifies what words a character should speak (with the use of a speech synthesizer).
- `<gesture>`: specifies coordinated movement with arms and hands.

For us, the most important behavior is

- `<face>`: movement of facial muscles to form certain expressions. Facilities exist for moving the eyebrows, mouth, eyelids but the core also specifies a place where FACS action units can reside.

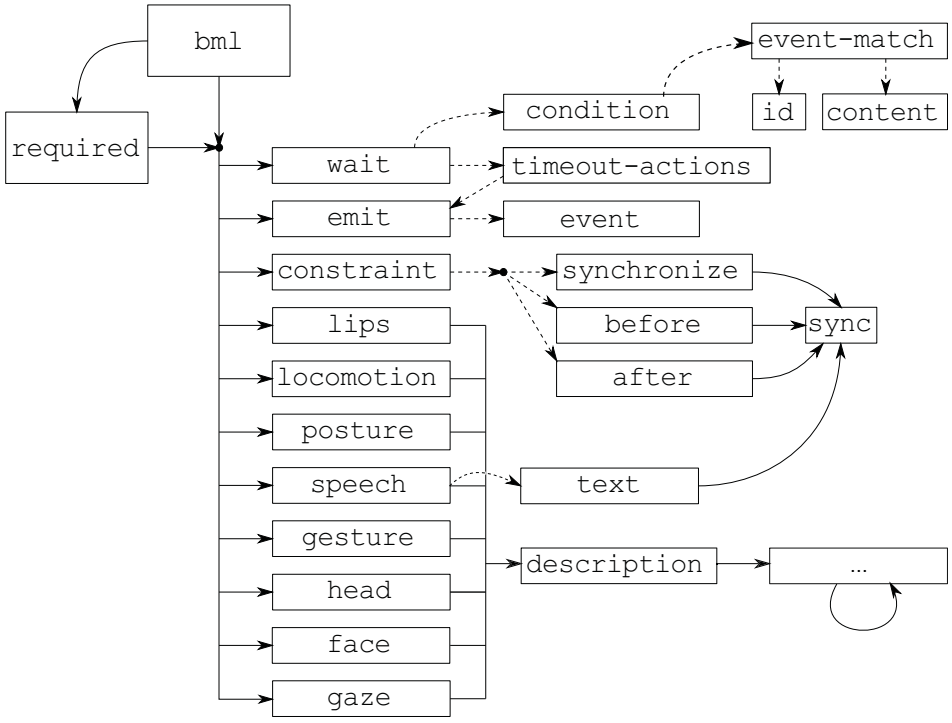


Figure 3.2: BML elements and its hierarchy. Each child in this graph can be a child node in XML. Dashed lines represent that the parent can only have one child of that type.

There is currently some discussion about what action units are required for a realizer to have implemented. For this project, we plan to support the full array of action units or only a beyond core specification of FAPs.

Besides these behavioural tags, several administrative elements are available for messaging, event synchronization and marking groups of behaviours as required. The full tree is depicted in Figure 3.2.

Behaviors have certain markers that are called synchronization points. These occur in time when the behavior is executed, mark beginning, stroke and end amongst others and can be used to synchronize other behaviors with these key moments. A behavior can be bound to a synchronization point of another behavior internally, but this also can be achieved by using dedicated tags externally to the behaviors.

3.2 Design

The design of the recursive descent parser is broken up in two pieces: the class hierarchy and the object hierarchy. Java is chosen as implementation language, mostly because it is used most of the time with current efforts at HMI.

Efforts are made to make the process of parsing a BML document reversible. This means that from a representation of the BML document in a Java object tree, the BML document can be reconstructed.

3.2.1 Class hierarchy

The basis of the recursive descent parser for reading BML is a recursive descent parser for XML. This can easily be extended for any kind of XML by extending the proper Java classes. The full class hierarchy can be found in Figure 3.3.

3.2.2 Class diagram

Within this class hierarchy, objects relate to each other. For example, a `RequiredBlock` can have zero or more multiple `Behaviors`. The full set of relations is depicted in the class diagram of Figure 3.4.

In this class space, a reference to an synchronization point is represented by an object of the type SyncRef. The classes Sync an Synchronize represent the tags `sync` and `synchronize`.

3.3 Scheduling

Some efforts have been made to design and build a basic scheduler for BML behaviors. In this section, the context of such a scheduler is described. To help determine in what situations the scheduler is needed in particular, some use cases are described. And for solving conflict situations, some possibilities for problem solving are given.

3.3.1 Context

Scheduling behaviors is done by the scheduler. It gets its BML from the planner, observer and event listener and outputs absolute timing information along with other BML encoded necessary information to the various engines. The scheduler also can query these engines for extra information. This could be preferred timings or some cost or penalty that this engines assigns to any given timing.

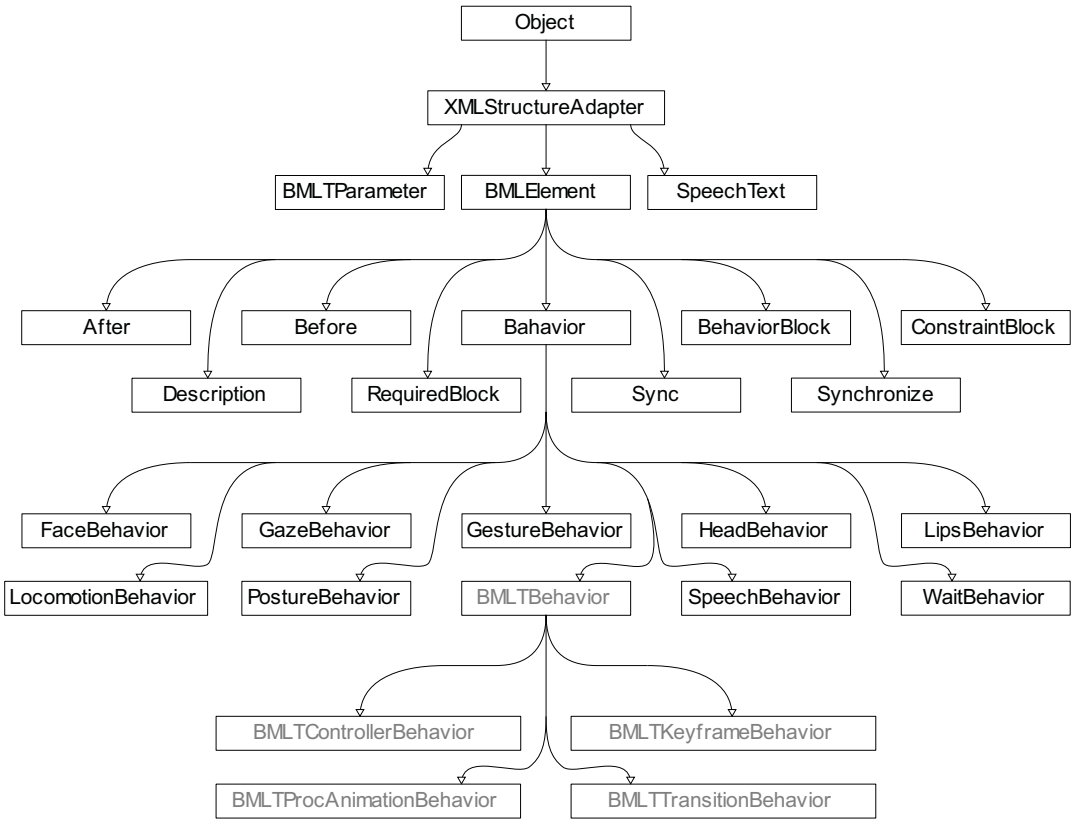


Figure 3.3: The Java class hierarchy. The classes with gray text represent an extension not designed by me.

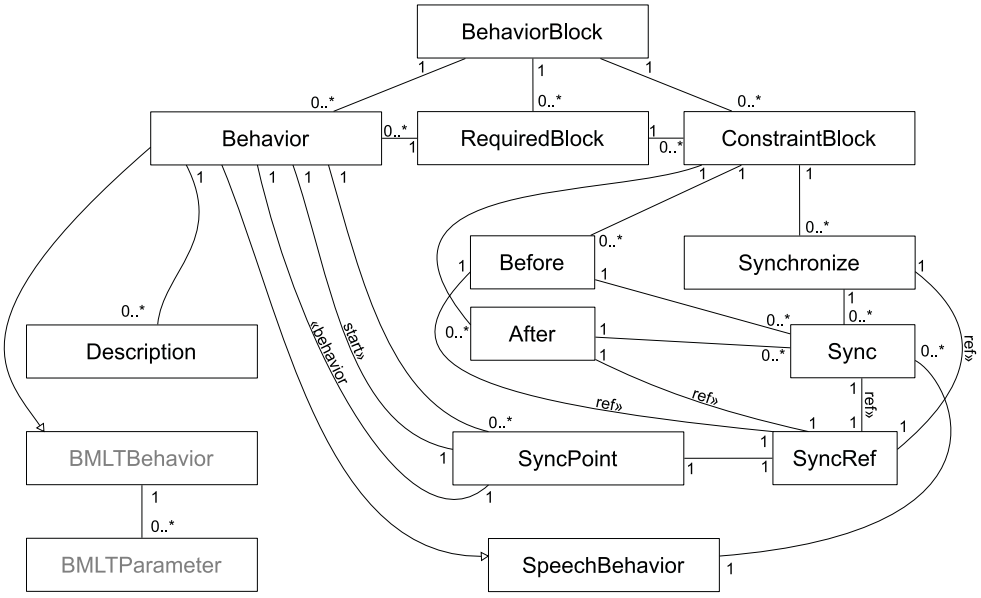


Figure 3.4: The class diagram. The classes with gray text represent an extension not designed by me.

The scheduler works continuous and parallel to the planner, observer, event listener and engines. Scheduling is all about lining up synchronization points. Every behavior can have one or more of these points and every behavior can reference each of these points to a specific point or area in time. A point in time is defined by means of a sync point of another behavior and a area in time by means of before or after a point in time.

As soon as a sync point is known to the scheduler and as long as referenced sync points are not consumed by time, they can be shifted. When none of the sync points of a behavior are consumed, the complete behavior may be shifted in time, without changing the individual sync points. Nothing fancy is there to be done here.

When two sync points of the same behavior are referenced to two sync points of another behavior, the penalty function provides a mechanism to find out what portion of time adjustment should go to each of the behaviors. The same holds for more behaviors that are interlocked with each other.

Some example scheduling problems. See Figure 3.5 for the most simple problem. Here, only one of the two need to be shifted - or translated - in time. Since if one behavior is inserted in the scheduler, it is executed as fast as possible, it is better to align left here too.

See Figure 3.6 for a scheduling problem where two synchronization points each point to the same behavior. The solution is to scale one of them or to scale both.

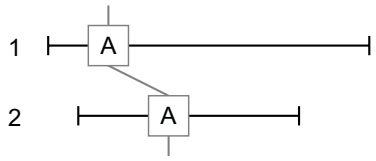


Figure 3.5: Shift one of the behaviors.

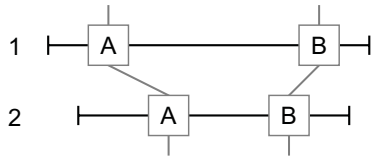


Figure 3.6: Scale one or both behaviors.

Figure 3.7 is a bit more complex. When the length of behavior 3 is altered to match up synchronization point C, this also has an effect on the target lengths of all other behaviors. Changing each of the behaviors has influence on all other.

3.3.2 Use cases

Consider the case of the virtual conductor. Imagine the conductor should show the tempo by using its arm and nodding its head. The behavior planner plans the behaviors accordingly and puts a synchronization constraint between the stroke of the arm movement and the stroke of the nods. One of the two modalities has to have authority because otherwise the behaviors are just shifted forwards in time when at least one of the elements of the animation (eg. the hand gesture or nod) is longer than the tempo-period. Nothing fancy is happening here since the scheduler has all synchronization points fixed in time.

In fact, most of the BML-scripts do not have scheduling problems in which there is no clear solution. But theoretically, for the cases that fall in the scheduling problems of figure 3.6 and 3.7, a solution is presented in the next section.

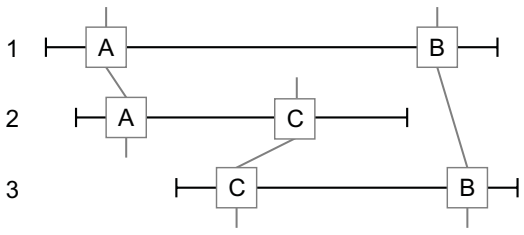


Figure 3.7: Scale one, a combination of or all behaviors.

3.3.3 Problem solving

Simple scale Consider the scheduling problem of figure 3.6. Simple scale means that both periods (between synchronization points A and B for both behaviors) are scaled to the mean of to their initial sizes. If period 1 has a length of 3 and period 2 a length of 2, the target length t of both periods is then $t = \frac{3+2}{2} = 2.5$

Quadratic cost functions Assume all behaviors have only one optimal length. This is the minimum of the cost function. It monotonically rises on both sides of this point. With these prerequisites, the balance between two or more behaviors can be found in a small amount of time.

Furthermore, if we restrict the functions to be quadratic in the form $y = a(t_1 - t_0)^2 + b(t_1 - t_0) + c$, we simply can add these functions up and lookup the minimum with $(t_1 - t_0) = \frac{-b}{2a}$.

Cost functions More complex cost functions that have multiple minimums and maximums are possible, but it is expected that those functions cannot help the scheduler find a optimal solution in bounded time in all cases.

3.3.4 SmartBody scheduler

SmartBody is a research project by the University of Southern California's Institute for Creative Technologies and Information Sciences Institute. It is a character animation system that uses BML to describe the movements a character needs to perform. The scheduler does adress translation or scaling of behaviors, but does this in the order the behaviors enter the system [24]. It does not address situations in which behaviors have a circular dependency, presumably because those situations hardly do occur.

3.3.5 Conclusion

The problems that the BML-scheduler faces can be divided in a few classes. It is to be expected that more complex cases are rare, considering use cases that the whole BML-realizer would be used in. But when complex cases must be processed, the polynomial cost functions are good candidates because they are more flexible than simply averaging the lengths of behaviors and still are easy to calculate.

Chapter 4

MPEG-4 Facial Animation

This chapter starts with a description of the main and most important collection of methods and techniques bundled in a standard called MPEG-4 Facial Animation (FA). After this, Xface - it is an open source implementation of MPEG-4 FA that aided in development and evaluation of our own implementation of the standard - is described. This section is followed by a description of the prototype itself and this chapter concludes with an evaluation of the quality of the prototype.

4.1 Standard

This section describes the MPEG-4 Facial Animation standard. It is part of MPEG-4 systems that has a characteristic producer consumer architecture with a one way transport link in between. Audio and video as well as Facial Animation (FA) and possibly others all have their own encoder and decoders at both ends of this link. The basic idea is to mark a face with a number of points (Feature Points, FPs). The position of these points and the vertices near them, is then controlled by parameters (Facial Action Parameters, FAPs). The distance of displacement is related to distances between key FPs.

FAPUs (Facial Action Parameter Units) are fractions of key-distances on the face. For example, the distance between the eyes. This allows usage of FAPs in normalized ranges so they are applicable to any model. See Figure 4.1. When a FAP has value 1024, its FP moves a distance equal to the corresponding key-distance.

FAPs (Facial Action Parameters) describe all possible actions that can be done with the face using MPEG-4 FA. This can either be done at low level by displacing a specific single point of the face or at a higher level by reproduction of a facial expression. Changing a FAP means changing the location of the corresponding FP, in relation to the appropriate FAPU and in the direction defined by the FAP itself. Table 4.1 gives some example FAPs.

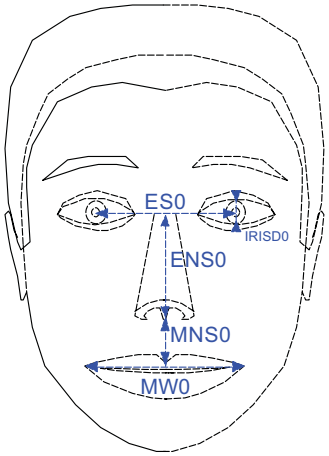


Figure 4.1: Facial Animation Parameter Units

Most FAPs are bidirectional and work in both directions (positive and negative) but some only accept positive values. Appendix B is a complete list of all FAPs.

FAP	Name	Description	Unit	Uni- / bidirectional	Motion
1	viseme	Set of values determining the mixture of two visemes for this frame (e.g. pbm, fv, th)			
2	expression	A set of values determining the mixture of two facial expression			
3	open_jaw	Vertical jaw displacement (does not affect mouth opening)	MNS	U	down
4	lower_t_midlip	Vertical top middle inner lip displacement	MNS	B	down
5	raise_b_midlip	Vertical bottom middle inner lip displacement	MNS	B	up

Table 4.1: FAPs. A full reference can be found in Appendix B.

FPs (Feature Points) are points on the face. Their position during expression or animation is altered by one or more FAPs. See Figure 4.2 for the position of all FPs in the face.

Now one might wonder, considering the low level topology of the face that consist of a

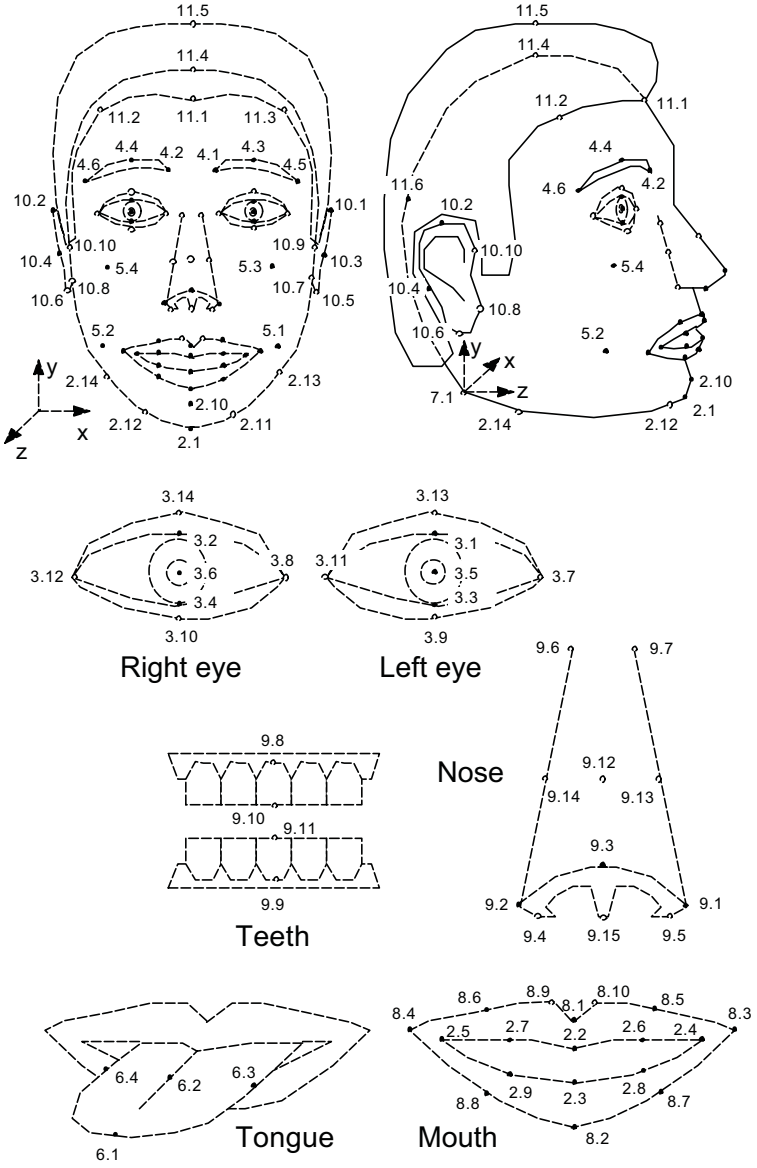


Figure 4.2: Feature Points. Solid dots actually represent points that can be controlled by FAPs.

number of vertices and some feature points on its surface who's position is altered, how do we change the position of these vertices in a way that we end up with a realistic looking face? There are several methods for this, some of which are described in 2.2.2.

4.2 Xface

4.2.1 Description

Because the first prototypes of conversions from FACS and emotion were built before our MPEG-4 FA implementation in Java, we had a need for a tool that could visualize a MPEG-4 FA stream. More than one was freely available on the internet, but Xface was chosen because of its unique ability to control it over TCP/IP. And this would come in handy when the conversion prototypes are to be fine-tuned because the actual face on the screen is updated almost instantly. See Figure 4.3 for a screenshot of Xface Player.

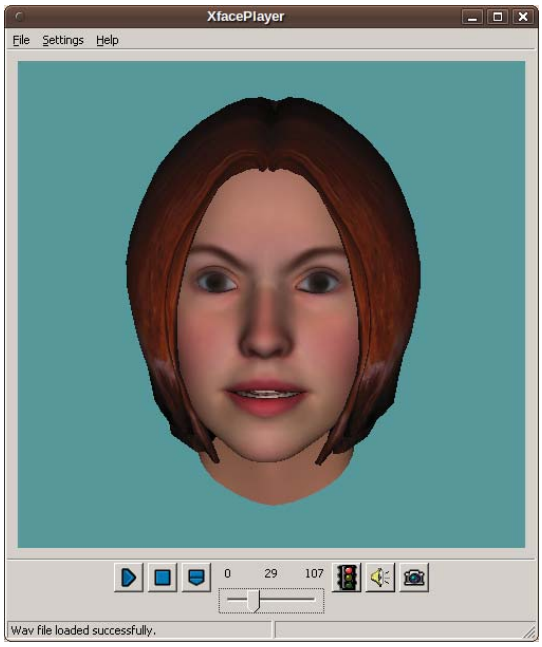


Figure 4.3: The Xface Player

4.2.2 Java-interface

To be able to use Xface throughout the whole project, a small part of the client side portion of the Xface TCP/IP protocol was implemented in Java. There was very little or no documentation except from what actually traveled over the line between Xface and its own client application and the source code. In addition, a few other problems arose which are described in the remainder of this section.

All actions that can be done via the network are called *tasks*. Xface reads a more or less common plain text file format for representation of FAP-values. The first line includes

the file-format version number, filename, speed (frames per second) and number of frames in the file. Per frame, two lines are used. The first line is a mask that tells the receiver for which FAPs values are supplied in the second line. The second line is also prepended with the frame number. The network protocol allows for giving a reference to such a file or directly uploading the contents.

The fact that Xface reads FAPs only per-file posed a problem because we wanted to display FAP-values in real-time without saving to a buffer first and showing it later. Getting Xface to accept files with only one frame has proven to be possible but two bugs in Xface that needed a workaround. First of all, Xface would not display a one framed file until a stop-commando was sent. And secondly, Xface would stop showing file uploads when they were sent over the line too quickly after each other. The solution to this last problem was to have new FAP values uploaded at only a 250 ms interval.

The actual `XfaceInterface` for Java includes a simple state machine, as shown in Figure 4.4. This is to keep track of our state and stops us from things such as connecting when connected, trying to communicate when no connection is open and for debugging purposes.

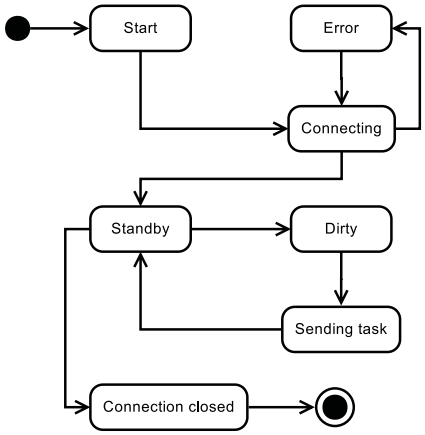


Figure 4.4: The simple state machine as used in the `XFaceInterface`.

4.3 Our MPEG-4 FA implementation

Our MPEG-4 FA implementation called FaceEditor is a Java application that loads and shows the head model, reads the file and provides the GUI for adjusting FP locations, setting and reviewing FAP parameters and interfaces with the prototypes for conversion from FACS and conversion from emotion. This prototype is described in the next few sections.

4.3.1 Software model

FaceEditor is build on top of the Elckerlyc environment, a 3D framework that handles the scenegraph, interfaces with OpenGL and loads objects. It can easily be extended as was done for FaceEditor. On top of this, various graphical user interface classes are created. See Figure 4.5 for a class diagram of the most important classes.

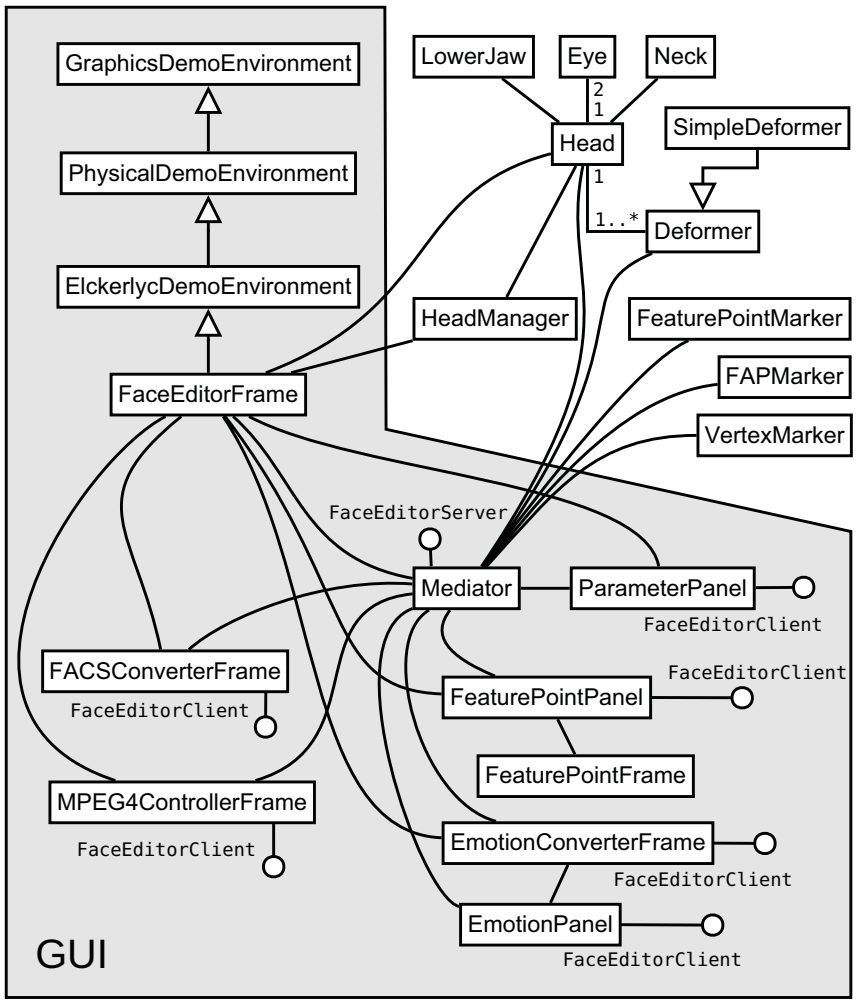


Figure 4.5: Most important Java classes used in FaceEditor

`FACSConverterFrame` and `EmotionConverterFrame` are entry points for respectively the FACS converter (see §5) and the emotion converter (see §6).

`FaceEditorFrame` is the class that overrides `ElckerlycDemoEnvironment`. When initiated, it starts by creating a new `HeadManager` which instantiates and returns an object

of the type `Head` which in turn handles loading of all parameters. After that, the GUI is constructed and the application is running.

The purpose of the classes `LowerJaw`, `Eye` and `Neck` is to encapsulate 3D world objects and provide an interface relevant to that object. For `LowerJaw` for example, FAPs 3 (open_jaw), 14 (thrust_jaw) and 15 (shift_jaw) can be set directly and the 3D world object is then positioned and rotated accordingly.

`Head` does not only have the task of loading and saving parameters, it also keeps track of displacements on a per-vertex basis. When displacements are to be applied to the 3D face mesh, displacements are averaged (when a vertex has more than one displacement) and set. Furthermore `Head` calculates FAPUs which are requested by objects of the type `Deformer`, `Eye` and `Neck`.

When handling GUI events, a lot of interaction is going on between the normal screen elements and the 3D world. The most important class that enables this interaction is the `Mediator`. It implements the interface `FaceEditorServer` which has methods for setting `MPEG4Configuration` objects. On the other side, many of the objects that interface with the `Mediator` implement the interface `FaceEditorClient` which has a method for passing the `Mediator` itself so they can communicate with it.

Furthermore, `Mediator` receives a lot of updates from GUI elements that let the user specify parameters for FAP-parameters. It translates these updates to appropriate actions to be taken on the `Head` object, `Deformer` objects and the 3D helper instruments `FeaturePointMarker`, `FAPMarker` and `VertexMarker`. `FeaturePointMarker` and `VertexMarker` are small boxes that show the positions of respectively FPs and vertices. `FAPMarker` is rendered as a wire-frame sphere that shows influence. More on this in §4.3.2.

The GUI is split in two important parts, one for setting locations of FPs and one for setting parameters of FAPs. The first part is handled by `FeaturePointPanel` and the second by `ParameterPanel`. `FeaturePointFrame` shows a reference image of where feature point should be placed on a face and can be opened from `FeaturePointPanel`.

4.3.2 GUI

In this section, the GUI of FaceEditor is described. Take a look at a screenshot of FaceEditor in Figure 4.6. The bar on the left is where FPs are selected and parameters are set. Auxiliary screens and functions can be found in the bar at the bottom of the screen. The main area on the right is where the 3D face and helper instruments are displayed.

Navigation through 3D space is inherited from the Elckerlyc environment. When focus is on the 3D portion of the window, keys can be used to move the camera. See Table 4.2 for an overview of these keys. Face and world orientations are aligned. The x-axis is pointing to the left (from our point of view, to the right for the point of view of the face), y-axis

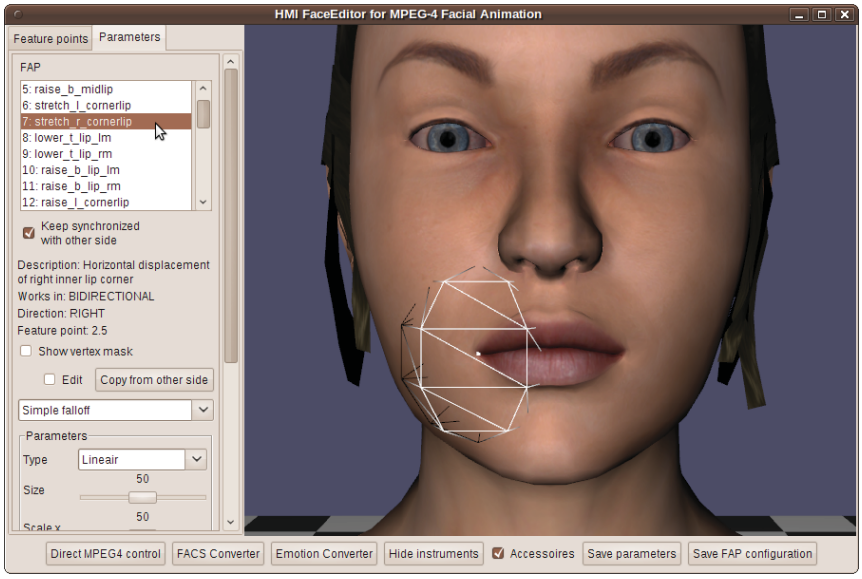


Figure 4.6: Screenshot of FaceEditor.

to above and z-axis to the front of the face.

Key	Action
Up	Move the camera forward
W	Move the camera forward fast
Down	Move the camera backward
S	Move the camera backward fast
Left	Turn the camera to the left
Right	Turn the camera to the right
Page-up	Move the camera up
Page-down	Move the camera down
A	Move the camera to the right
D	Move the camera to the right

Table 4.2: Navigation keys, expressed in terms of orientation of the camera.

A few actions require a translation from a location from the 2D panel that shows the rendered 3D world to a 3D coordinate. When a 2D coordinate is known, the z-depth of this location is retrieved and with a few matrices for projection and viewport matrices, the 3D coordinate is calculated.

When setting parameters for a face for the first time, positions of feature points, see Figure 4.2, must be set first. The process is very simple. When clicking the 3D panel, the position

of the currently selected feature point is set to the 3D coordinate where this click occurred and a marker is also placed at this location. When another feature point that already has a position is selected, the marker is moved to this location so they can be reviewed and reset when needed. See Figure 4.7 for two relevant portions of the GUI.

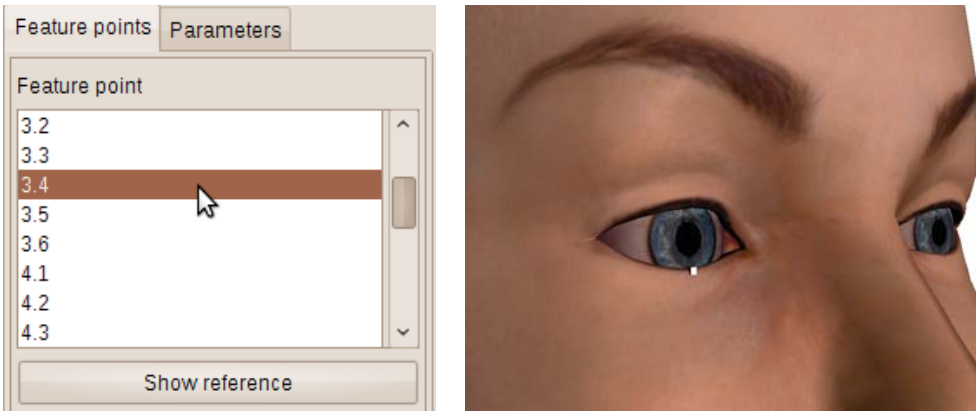


Figure 4.7: The GUI for selection of FPs and the marker indicating current FP positions.

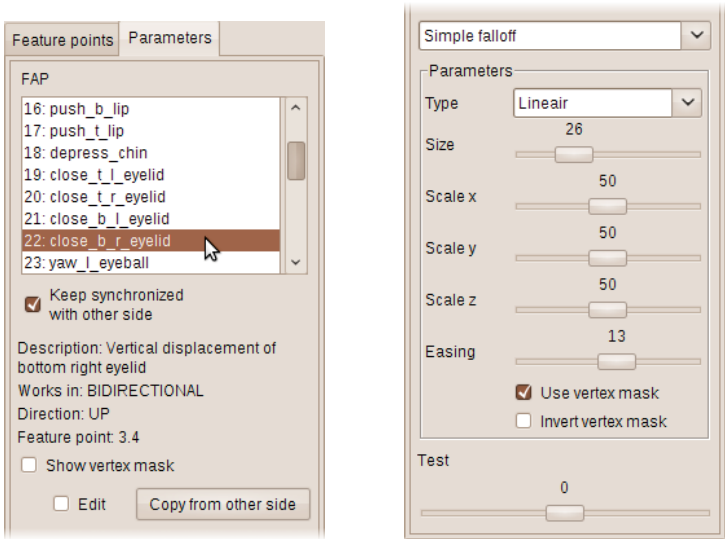


Figure 4.8: The parameter panel.

Now that the locations of feature points are correct, parameters can be set. The parameter panel can be split in three parts; the FAP selection and information-part, the parameter part and the test part. See Figure 4.8 for how this looks.

The first part gives a list of all available FAPs. When a FAP is selected, the rest of the panel is updated, the feature point marker is moved to the location of the currently relevant feature point and the FAP marker is moved and sized according to the actual size and shape of the influence sphere.

Keep synchronized with other side is only enabled when a FAP is selected that has a counterpart on the other side of the face. close_b_r_eyelid and close_b_l_eyelid for example. When it is checked, any subsequent changes in all parameters are also made to the FAP of the other side.

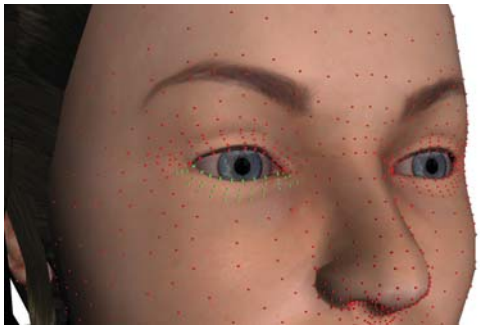


Figure 4.9: The vertex mask showing selected vertices for FAP 22 (close_b_r_eyelid).

The basis of facial expression in FaceEditor is the displacement of feature points and the points or vertices that surround them. Since we select those vertices based on the distance from the feature point, in some cases some vertices are displaced when we do not want them to change position. For these cases, it has been made possible to individually select vertices and create a vertex mask. See Figure 4.9 for how this looks. Vertices can be selected or deselected by clicking the face whenever the checkboxes *Show vertex mask* and *Edit* are checked. The vertex marker closest to where the click was made, is selected or deselected. The button *Copy from other side* attempts to copy the vertex mask from the other side. For this, vertices on the left and on the right side of the face are required to be symmetrical about the vertical plane in the middle of the face within a small margin. See §4.3.3 for more on the effect of vertex masks.

The second part lets the user choose between several types of vertex displacement. The two shown here, simple falloff and linear (Figure 4.8), are until now the only ones available because the simple falloff combined with xyz-scaling, easing and vertex masks seem sufficient for now. The sliders for size, scale x, scale y and scale z specify the size and shape of the sphere of influence and easing influences the rate of falloff as function of the distance to the feature point. See §4.3.3 for more on easing.

While setting parameters, the user can test current parameter values by moving the test slider. All subsequent changes to parameters and the vertex mask are shown directly. See Figure 4.10.

Figure 4.10: The feature point marker and FAP marker while the test slider is in neutral position and while it is set to 600.

A more elaborate way of testing parameters is by using the *MPEG-4 controller* utility which can be started using the corresponding button on the bottom bar of FaceEditor. Values can be set on a per-FAP basis and reviewed instantly in FaceEditor. See Figure 4.11 for a screenshot.

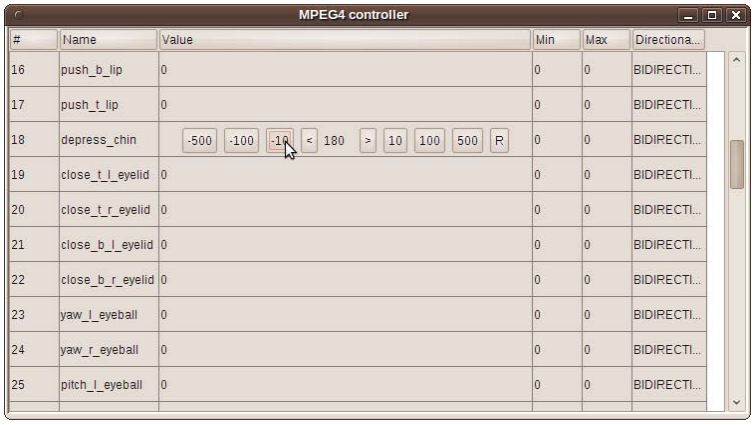


Figure 4.11: Screenshot of the MPEG-4 direct control utility.

Other buttons on the bottom bar are respectively for the FACS converter (see §5), for the emotion converter (see §6), to hide the instruments such as the markers, show or hide accessories such as eyes and teeth, to save the parameters to a new XML file and to save the current FAP configuration to a FAP file.

4.3.3 Displacing vertices

According to the value of a FAP, the corresponding feature point is moved in the direction specified by the standard. Vertices surrounding the feature point are for now always moved in the same direction.

The distance each vertex moves is related to its distance to the feature point and the

radius of the influence sphere which center is also at the feature point. The function in which we can describe this behavior travels from 1 when the distance is 0 to 0 when the distance is equal to or larger than the radius of the influence sphere. The outcome is the influence, or i. This function is linear in the normal case, but there are a few mechanisms that can influence the distance a vertex is displaced apart from sizing the influence sphere: scaling, easing and masking.

Scaling is the process of changing the size of the influence sphere in only one or two dimension so that it becomes an ellipsoid or, when two dimensions are scaled equally, a spheroid. The aforementioned function then travels to 0 when the vertex approaches the end of the imaginary line through the vertex and between the center (the feature point) and the surface of the shape.

Easing is the process of changing the influence curve by exponentiation. Normally, $i' = i$. Easing can be done in two directions: by easing in and increase influence or by easing out and decrease influence. The GUI allows for an input of the ease parameter e ranging from -100 (easing out) to 100 (easing in). When easing out, the exponent is $1 + e/100$ and when easing in, the exponent is $1 + e/20$. See Figure 4.12 for a number of curves for various values of e.

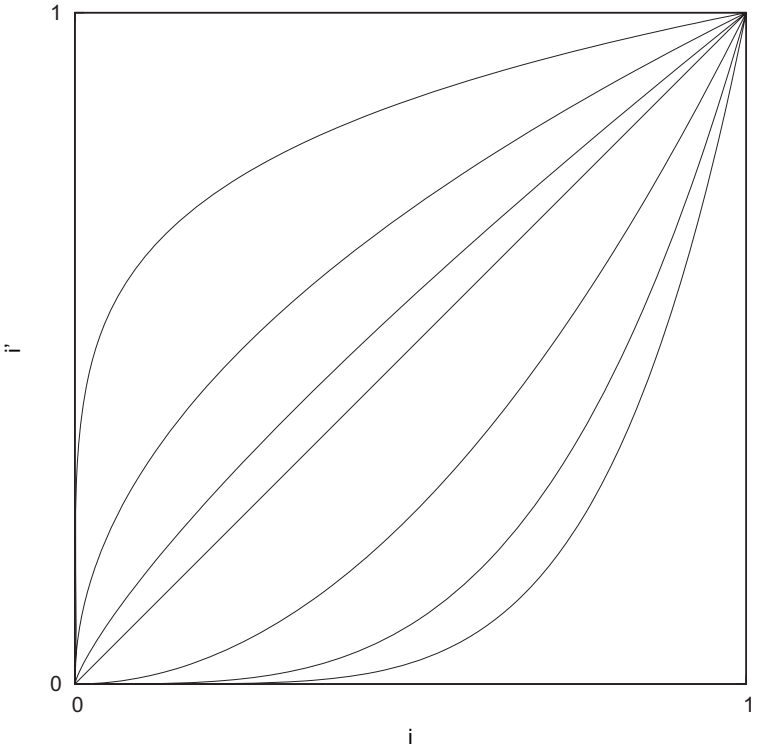


Figure 4.12: Some sample curves that are used for altering the influence curve. From top to bottom $e = -80, e = -50, e = -20, e = 0, e = 20, e = 50, e = 80$.

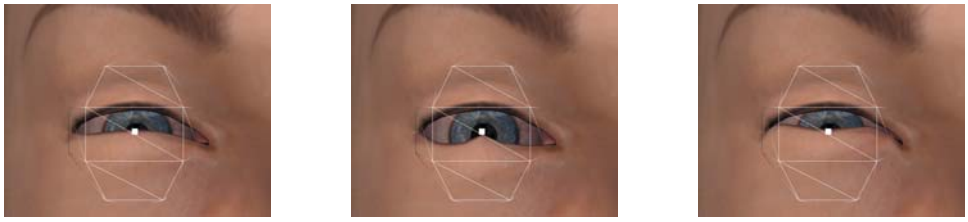


Figure 4.13: Easing for the right lower eye lid. From left to right: normal situation, easing out and easing in.

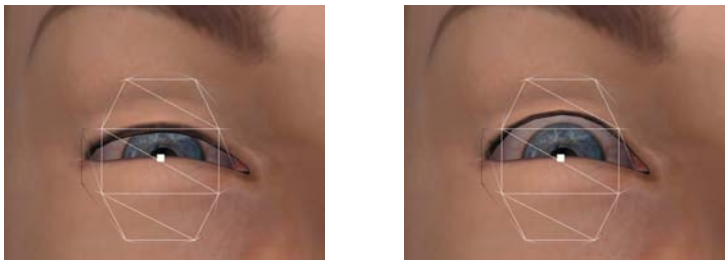


Figure 4.14: Masking for the right lower eye lid. In the right image, the vertex mask is disabled and the upper eye lid moves along.

Masking makes it possible to switch off displacement on a per-vertex basis. This is necessary in cases where parts that should move are close to parts that should not. Eyes and the mouth are good examples of this.

There are several things that can be changed in this process. More sophisticated implementations may change the direction in which vertices travel based on its relative position to the feature point to get a more real muscle-based contraction. Also, the whole idea of the influence sphere, easing and masking is to assign weights to vertices. Numerous other procedures can be followed here, such as vertex weight painting, defining regions, etcetera.

4.3.4 Alternatives to easing

We were concerned about the fact that for all of the curves produced for easing, the first derivative is never equal to zero for $i = 0$ and $i = 1$. When displacing vertices in a mesh that consists of an infinite number of points, side-effects could become visible. A few possible solutions (for making the first derivative through $i = 0$ and $i = 1$ equal to zero while maintaining smoothness of the curve) have been put to the test.

First, over a small interval at the beginning and the end, such as $[0.0 : 0.2]$ and $[0.8 : 1.0]$, the original easing curve was adjusted using a hyperbolic tangent. See Figure 4.15 for a plot of these curves. The problem with this is that the first derivative gets to large at

times where the curve needs to *catch up* a lot in order to maintain a smooth descent of the derivative to 0 at $i = 0$ for example.

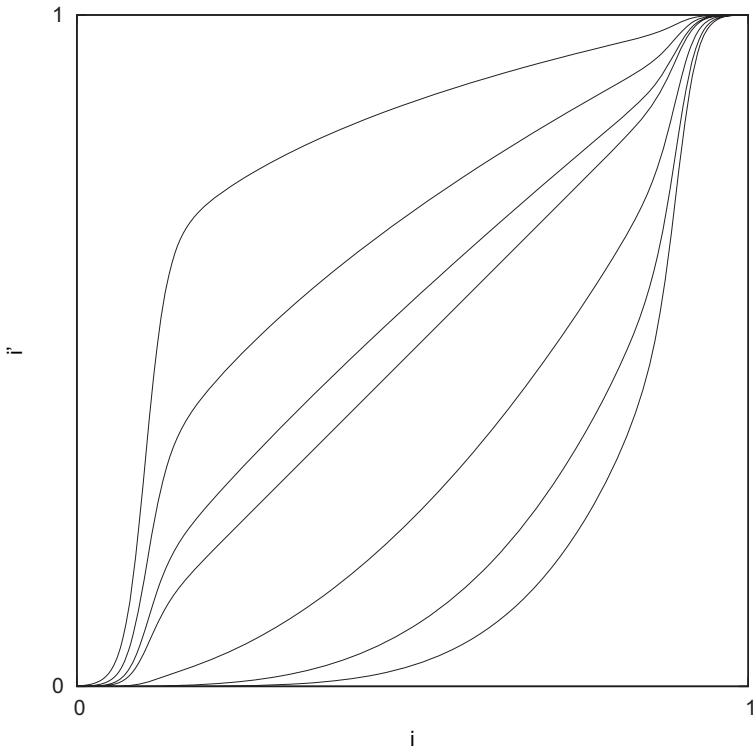


Figure 4.15: Some sample curves that are used for altering the influence curve and that are smoothed with a hyperbolic tangent. From top to bottom $e = -80, e = -50, e = -20, e = 0, e = 20, e = 50, e = 80$.

Secondly, easing was ignored and replaced by a combination of Bézier curves and automatic adjustment of the size of the influence sphere. Placement of control points is determined by two new parameters, smooth center and smooth side. See Figure 4.16 for a plot of the curves where smooth center and smooth side are on 0%, 25%, 50%, 75% and 100% of the size of the original influence sphere. The right side of the curve always corresponds to the center of the influence sphere and the left side always corresponds to the edge of this sphere.

Ignoring easing and replacing it with smoothing using Bézier curves yields very slightly different results, but it's mileage may increase when models with even more vertices are introduced.

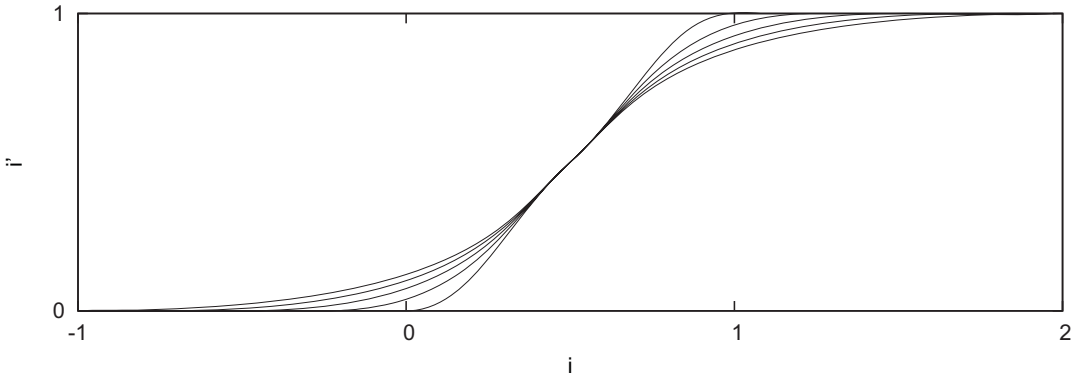


Figure 4.16: Some sample curves that are used for altering the influence curve based on Bézier curves and enlarging the actual sphere of influence.

4.3.5 Setting parameters for a new face

A set of parameters must be set first in order to have face show expressions. This process all can be done from within the GUI of FaceEditor and the process that needs to be followed is outlined shortly in this section.

- Place all feature points on the face using the example image from the standard.
- Set parameters for each of the FAPs:
  - Set the size of the influence sphere appropriate to the feature point and surrounding feature points. For points next to each other such as on the eyelids, a good default is to adjust the radius of the influence sphere so that it just includes the neighboring feature point.
  - Use masking for the lips since for lower lip movements, the upper lip vertices must stay in their positions and vice versa. It might be a bit cumbersome to grab the right vertices when they are hidden, but activating the fap during vertex selection might make things easier.
  - Easing comes in handy with things such as the midpoints of the eyelids. The vertices in the middle between these midpoints and the corners of the eye would normally not move enough, easing in can adjust for this.
  - Alternative to easing, smoothing can be used to smooth out the influence near the center or near the edges of the influence sphere.
  - Some FAPs such as the jaw lowerer, need to have the influence sphere flattened because we don't want the vertices above the lower lips to be influenced although we do want to cover the whole width of the face. This is done with the x-, y-, and/or z-scaling.

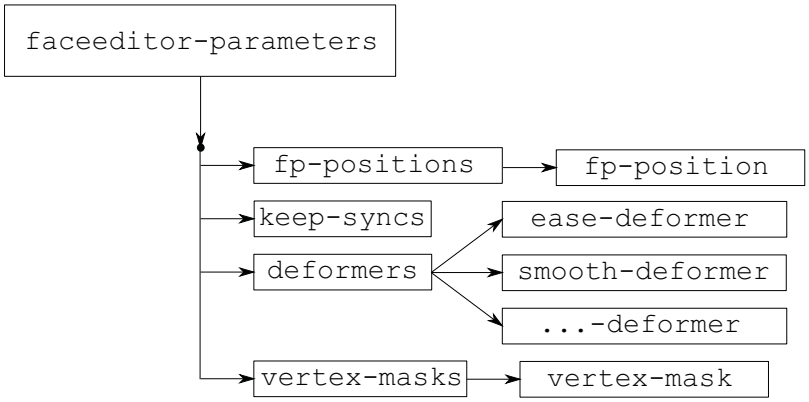


Figure 4.17: The XML file format.

– We found that when parameters for individual FAPs are set for the first time, only a few adjustments are yet to be made before activating FAPs cooperatively. So regularly test the FAP using the test slider.

4.3.6 File format

When parameters are set, they must be saved to a file to make them persistent and available when FaceEditor is run the next time. The storage format has changed two times over time. First, native Java object serialization was used. This can be implemented very quickly and is integrated in Java and the Java code. Attributes can be kept from serialization by the `transient` keyword and no extra code is needed because everything is happening under the hood. Drawback is the fact that the file format cannot be read and altered by humans directly using a plain text editor. Flattened objects are not bothered by adding or removing attributes, as long as `serialVersionUID` is used, but is is not possible to change a object's hierarchy.

To overcome the issue that the file is not human readable, a simple plain text file format was incorporated. Positions of feature points, parameters of FAPs and vertex masks were written to a file as simple as possible. The drawback of this is that when these parameters need to be embedded in some other file, chances are that the exact contents of the file cannot be kept intact. This was solved by the use of XML.

The hierarchy used is simple and plain, see Figure 4.17 and Appendix C for a DTD and a textual description.

Figure 4.18: The faces used in evaluation of FaceEditor. From left to right: Xface, Greta and Miraface.

4.4 Evaluation

4.4.1 Faces

The faces that are being used in this evaluation are Xface, Greta and Miraface which are described in this section.

Xface is already shortly described in 4.2.2. The Xface project is initiated and maintained by the Cognitive and Communication Technologies (TCC) division of FBK-irst, a research center based in Italy. It is open source and platform independent [5]. See Figure 4.18 for a screenshot.

Greta is a "Simple Facial Animation Engine (SFAE)" which aim was to have "an animated model able to simulate in a rapid and believable manner the dynamics aspects of the human face". It includes the ability to generate wrinkles using the bump mapping technique [20]. See Figure 4.18 for a screenshot.

Miraface is facial animation software. It incorporates a facial animation module and includes a simple facial model both developed at MIRALab and has a relatively low number of polygons. See Figure 4.18 for a screenshot.

Attempts have been made to also use software supplied by Visage Technologies AB, visage—interactive, but those failed. The key of the problem lies in the fact that this program is only able to read a binary stream of FAPs that are encoded in the binary MPEG-4 FBA data stream and that conversion is very labor intensive. Furthermore, actual visualisation still showed to be unrealistic. See Figure 4.19 for a screenshot of visage—interactive and how a certain FAP configuration was visualized.

RUTH is also an animatable face, see DeCarlo et al. [7], but it only has some mouth and tongue movements along with brow actions, smiling and blinking so it is not MPEG-4 FA

Figure 4.19: From left to right: visage—interactive with model *Reana* loaded, how a certain FAP configuration looks on Reana and how this same configuration should look using FaceEditor. This FAP configuration was actually created using the Emotion conversion prototype described in §6.

compatible.

4.4.2 Method

Directly comparing displacements of the face in FaceEditor with displacements of other faces that have implemented MPEG-4 FA, for the same FAP-values, is a way to evaluate the quality of the implementation of MPEG-4 FA and the parameters that are set. Although still subjective, it is possible to compare displacements and determine whether they are similar or whether there is a displacement that is better (more realistic) than the other. Attempts to this are described in this section.

Note that only individual FAPs are evaluated in this section. We do not assume that when activation of all individual FAPs are looking realistic, combinations are too. A more high level evaluation is performed in §7, also incorporating the more high level steering methods. Also, we evaluated FaceEditor without using any smoothing as described in §4.3.5.

There is a bottle neck in the evaluation on this level. A ground truth only exists for displacements of feature points. MPEG-4 FA does not describe how vertices are best displaced, in particular because there is a infinite number of three dimensional face models. Because of this, determining realism of a displacement and comparing two different faces to obtain the most realistic one is a subjective human process. On top of that, the face model itself also has a influence on the actual quality of a displacement.

In defence, we are comparing FaceEditor not only with Xface but also with Greta and Miraface (see §4.4.1). The average of the displacements of all of these faces should at least approach a common ground truth. And since the face is a very important interface to humans, assessment of what displacement is more realistic should be quite universal. And

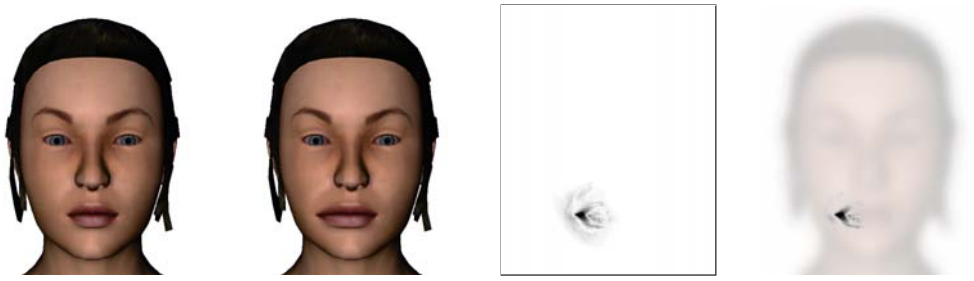


Figure 4.20: Creation of a difference-comparison image. From left to right: neutral face, FAP 7 (stretch_r_cornerlip) activated (in positive direction), the difference between these images and the difference with a blurred underlay.

regarding the influence of face models themselves, attempts are made to ignore them.

Screenshots were taken from the side when the displacement can not be seen very well from the front. This is the case for FAP 14 (thrust_jaw) for example.

The process of evaluation consists of creating screenshots of all faces for all FAPs. Most of the time, the value chosen is approximately so that the feature point moves halfway the FAPU. For bidirectional FAPs, another screenshot is taken with its value negated. Screenshots are cropped and the background is masked out so the only thing left is the face itself. Since differences between a certain pose and the neutral face are sometimes hard to spot, difference images have been calculated. Since these difference images only have differences in them, it is sometimes hard to determine where and to what extent exactly this difference in the face occurred. For this, the image of the neutral face (with the background removed) was blurred and placed with 25% intensity as layer under the differences. See Figure 4.20 for a visual display of this process.

Screenshots and difference images with blurred underlays can be viewed next to each other so that for all FAPs, the displacements of all four faces on these FAPs can easily be assessed in relation to each other. I assigned a score to each displacement, and for bidirectional FAPs, one for each direction. When a FAP (or direction) is not implemented, no score is given. Because FAPs vary in importance, each FAP is given a weight factor w from 1-3, 1 meaning not important (such as eyeball thrust), 2 meaning average importance (such as sub-lip displacements) and 3 meaning important (eyebrows, eyelids, mouth corners). In the end, a weighted average is calculated for each face with and without consideration of displacements that are not implemented.

These are 66 FAPs (the first two high level FAPs are left out), out of which 5 are unidirectional. So there is a total of $61 + 66 = 127$ displacements to be evaluated.

4.4.3 Analysis

See Table 4.3 for all scores given to the displacements. Some simple statistics can be found in Table 4.4. The weighted averages can be found in Table 4.5. One score is calculated while leaving unimplemented FAPs out of the equation and one while giving all unimplemented FAPs a score of 0 (to make it harder for implementations that only implement a few FAPs).

The critera for certain scores are:

- For a score of 3: the displacement is all right and looks the way it should (given the description of the FAP).
- For a score of 2: the displacement looks all right on first sight but is slightly odd (wrong displacement distance or an influence area that has a unrealistic size).
- For a score of 1: by the location of the displacement, it should be possible to reconstruct what FAP was activated.

It goes beyond the scope of this document to comment on all scores individually. However, an external document has been created which shows all screenshots side by side annotated by what is wrong with a certain displacement and why a certain score has been chosen, see Paul [2]. There are however some general remarks to be made here.

- Xface was particularly bad in displacements of eyelids and lip corners.
- Greta got left and right confused for FAPs 10 (raise_b_lip_lm) and 11 (raise_b_lip_rm).
- In Miraface, all FAPs working on the right half of the face work on the left half instead, and vice versa. This is consistent for all FAPs that have a counterpart on the other side of the face, so no scores were lowered for this.
- In some occasion, Miraface did not show any displacement for a right half FAP though it did for the left half FAP.
- For a certain number of FAPs, Miraface had no differentiated displacement for left and right and just showed the same symmetrical one for both FAPs.

See Figures 4.21, 4.22 and 4.23 for some example displacements and how they were scored.

There is little room for improvement for FaceEditor since there are only 13 displacements that not have been assigned a score of 3 (the gray line bordered cells in Table 4.3). These are for lowering and raising the corners of the mouth, lowering the midpoint of the top lip and stretching the nose. With the current implementation, it should be relatively easy to correct the displacements of the corners of the mouth. The lowering of the top lip

		Negative				Positive				
F	Description	FE	XF	GR	MF	FE	XF	GR	MF	*w*
3	open_jaw					3	3	2	2	3
4	lower_t_midlip	3	2	3	3	2	2	2	3	1
5	raise_b_midlip	3	1	3	3	3	1	1	3	1
6	stretch_l_cornerlip	3	3	1	3	3	3	1	3	2
7	stretch_r_cornerlip	3	3	1	3	3	3	1	3	2
8	lower_t_lip_lm	3	3	3	1	3	2	3	1	2
9	lower_t_lip_rm	3	3	3	1	3	2	3	1	2
10	raise_b_lip_lm	3	2	2	1	3	2	2	1	2
11	raise_b_lip_rm	3	1	2	1	3	2	1	1	2
12	raise_l_cornerlip	2	1		3	2	1		3	2
13	raise_r_cornerlip	2	1	2	3	2	1	2	3	2
14	thrust_jaw					3		3	3	1
15	shift_jaw	3		2	3	3		2	3	2
16	push_b_lip	3	2	3	1	3	2	3	1	2
17	push_t_lip	3	2	3	1	3	2	3	1	3
18	depress_chin	3		2	2	3		3	2	2
19	close_t_l_eyelid	3	1	3		3	3	3	3	3
20	close_t_r_eyelid	3	1	3		3	3	3	3	3
21	close_b_l_eyelid	3	1	3	3	3	1	3	3	3
22	close_b_r_eyelid	3	1	3	3	3		3	3	3
23	yaw_l_eyeball	3		2	0	3		2	0	2
24	yaw_r_eyeball	3		2	0	3		2	0	2
25	pitch_l_eyeball	3		2	0	3		2	0	2
26	pitch_r_eyeball	3		2	0	3		2	0	2
27	thrust_l_eyeball	3			2	3			2	1
28	thrust_r_eyeball	3			2	3			2	1
29	dilate_l_pupil				3				3	1
30	dilate_r_pupil				3				3	1
31	raise_l_i_eyebrow	3	2	3	3	3	3	3	3	3
32	raise_r_i_eyebrow	3	2	3	3	3	3	3	3	3
33	raise_l_m_eyebrow	3	2	3	2	3	2	3	2	3
34	raise_r_m_eyebrow	3	2	3	2	3	2	3	2	3
35	raise_l_o_eyebrow	3		3	2	3		3	2	1
36	raise_r_o_eyebrow	3		3	2	3		3	2	1

		Negative				Positive				
F	Description	FE	XF	GR	MF	FE	XF	GR	MF	*w*
37	squeeze_l_eyebrow	3	3	3		3	3	3	1	3
38	squeeze_r_eyebrow	3	3	3		3	3	3	3	3
39	puff_l_cheek	3		3	3	3		3	3	2
40	puff_r_cheek	3		3		3		3		2
41	lift_l_cheek					3			3	2
42	lift_r_cheek					3			3	2
43	shift_tongue_tip				3				3	1
44	raise_tongue_tip							1		1
45	thrust_tongue_tip									1
46	raise_tongue			1	3			1	3	1
47	tongue_roll							1		1
48	head_pitch		2	3	2		2	3	2	1
49	head_yaw		3	3	2		3	3	2	1
50	head_roll		2	3	2		2	3	2	1
51	lower_t_midlip_o	3	2	2	2	3	2	3	3	2
52	raise_b_midlip_o	3	1	2	2	3	1	2	3	2
53	stretch_l_cornerlip_o	3	2	3	3	3	2	3	3	2
54	stretch_r_cornerlip_o	3	2	3	3	3	2	3	3	2
55	lower_t_lip_lm_o	3	2	2	1	3	3	2	1	2
56	lower_t_lip_rm_o	3	2	2	1	3	3	2	1	2
57	raise_b_lip_lm_o	3	2	1	1	3	1	1	1	2
58	raise_b_lip_rm_o	3	1	1	1	3	1	1	1	2
59	raise_l_cornerlip_o	2	1	3	3	2	1	3	3	3
60	raise_r_cornerlip_o	2	1	3	3	2	1	3	3	3
61	stretch_l_nose	2		3	3	2		3	3	1
62	stretch_r_nose	2		3	3	2		3	3	1
63	raise_nose	3		3		3		3	2	1
64	bend_nose	3		3	3	3		3	3	1
65	raise_l_ear	3		3	3	3		3	3	1
66	raise_r_ear	3		3	3	3		3	3	1
67	pull_l_ear			3	3			3	3	1
68	pull_r_ear			3	3			3	3	1

Table 4.3: Scores given to all individual FAP movements to all faces. For bidirectional FAPs, scores are given for resp. the negative and the positive value. FE, XF, GR and MF are short for FaceEditor, Xface, Greta and Miraface. w is the weight of the FAP as explained in §4.4.2. Gray line bordered cells are the situations in which FaceEditor performs worse than at least one of the other faces.

	FaceEditor	Xface	Greta	Miraface
Implemented	104	70	110	115
Score 3	91	19	71	62
Score 2	13	30	25	24
Score 1	0	21	14	21
Score 0	0	0	0	8

Table 4.4: Number of implemented FAPs and number of FAPs that has been given a certain score. A total of 127 displacement have been assessed.

	FaceEditor	Xface	Greta	Miraface
Score without unimplemented FAPs	2.88	1.97	2.55	2.17
Score with unimplemented FAPs	2.60	1.32	2.32	1.96

Table 4.5: Weighted average score for each of the faces, with or without consideration of unimplemented FAPs.

Figure 4.21: Example of displacements for FAP 8 (lower_t_lip_lm) in the negative direction. All faces except Miraface has been given a score of 3 for this particular displacement. As can be seen, Miraface shows a symmetrical displacement here, which resulted in a score of 1.

Figure 4.22: Example of displacements for FAP 20 (close_t_r_eyelid) in the positive direction. Here, all faces got a score of 3.

Figure 4.23: Some examples of displacements of Xface that are considered bad and that are given a score of 1. From left to right, these displacements belong to FAPs 12 (raise_l_cornerlip, in negative direction, sharp edges and auxiliary displacement on other side), 19 (close_t_l_eyelid, in negative direction, strange black area above the eye), 22 (close_b_r_eyelid, in negative direction, hardly any movement at all) and 52 (raise_b_midlip_o, in positive direction, very unrealistic due to sharp corners).

Figure 4.24: Some displacements on which FaceEditor did not get the highest score. From left to right, these displacements belong to FAP 4 (lower_t_midlip, in positive direction, sharp point), 13 (raise_r_cornerlip, in positive direction, no curve in lip line) and two belong to FAP 61 (stretch_l_nose, in both directions, slightly unrealistic).

could pose a problem since adjusting this will also alter the negative version. Altering parameters to improve stretching of the nose will be harder because for this, vertices should travel in a angular fashion away from the center of the nostril. This is currently not possible because vertices are bound to the direction of the feature point itself. But this is only a minor problem because these FAPs for stretching the nose (and lowering the midpoint of the top lip) are not so important. See Figurer 4.24 for some of these cases in which FaceEditor did not get the highest score.

4.4.4 Conclusion

Several other MPEG-4 FA compliant faces have been used to evaluate the performance of FaceEditor itself: Xface, Greta and Miraface. Attempts have been made to also use

software of visage in order to have a fifth face, but those failed.

Although evaluation is inherently subjective, the method proposed at least attempts to create a metric whose determination and calculation procedures are more or less objectively reproducible. It does this by scoring individual displacements in the faces for each FAP and with each FAP having a weight indicating its importance, a weighted average is calculated. Using this method, FaceEditor scores better than all faces that were used in this evaluation.

Chapter 5

Conversion from FACS

Expressions that can be made by the human face were used to create a mapping that accomodates automatic and real-time translation from FACS to MPEG-4 FA. A software prototype was created for testing the mapping. This chapter describes the procedure used when creating the mapping, the software prototype and the evaluation performed.

5.1 Procedure

This section is all about how the translation between AUs and FAPs was constructed. The primary input was the mapping given by Zhang et al. [27], see Figure 2.3 and a set of images visually showing the impact of the activation of an AU which, courtesy of the Carnegie Mellon University School of Computer Science [17]. With the Xface Java-interface (see §4.2.2) and the MPEG4-direct control utility (see §4.3.2), the following steps were followed. For each given mapping (a AU-FAP-pair):

- Determine the maximum value by looking at the face produced by Xface. Any higher values should produce an unrealistic displacement of the feature point. We assume that Xface correctly performs the calculation of this displacement based on FAPUs of the face used.
- Determine, based on the description of the AUs and the muscles used, whether there are FAPs other than those assigned by Zhang et al. that positively can contribute to the appearance of the movement. For example, for AU 1 (inner brow raiser), FAP 33 (vertical displacement of left middle eyebrow) was added. See Figure 5.1 for a comparison.
- When more FAPs are mapped to this AU: possibly customize the start point.


Figure 5.1: Two screenshots of Xface: the right image has FAP 33 added to the mapping of AU 1, which originally only referred to FAP 31.

All FAPs mapped to an AU have two ranges each of which has a lower and an upper boundary. One range is for the FAP and one for the AU. Conversion from the one into the other is a simple linear calculation.

5.2 Our FACS conversion implementation

This implementation consists of:

- a file format for representation of the mapping between AUs and FAPs;
- a Java model to represent AUs and FAPs
- a Java application that reads a mapping file, supplies the user with a GUI with sliders corresponding to the AUs and interfaces with XFace and FaceEditor.

The file format is simple and plain. It it text based and each significant line (comments and empty lines are not) can either be a reference to an AU or a specification of the influence on a particular FAP. Each FAP influence specification is related to the last encountered AU reference. Since AUs do not specify whether a movement should occur on the left or right side of the face and FAPs do, we should make a distinction for FAPs whether they relate to a left or right side AU movement. This is done by giving the lines with AU references another field (separated with a tab) that is either 'L' or 'R' for left and right respectively. A FAP influence specification contains five fields: the FAP-number, AU-range start, AU-range end, FAP-range start and FAP-range end.

See Figure 5.2 for a few lines from a file in the format described above. AU 23 (lip tightner) has two counterparts in MPEG-4 FA, a left and a right one. The FAP for the movement in the left half of the face, FAP 53 (vertical displacement of left outer lip corner), ranges

from 0 to −300 when AU 23 ranges from 0.0 to 1.0. When the AU is activated half way down its range (0.5), the FAP is calculated as $0 + (-300 - 0) * 0.5 = -150$.

```
23 L
53 0.0 1.0 0 -300

23 R
54 0.0 1.0 0 -300

24
16 0.0 1.0 0 1000
17 0.0 1.0 0 500

25
3 0.0 1.0 0 900
4 0.5 1.0 0 -100
5 0.0 1.0 0 -1000
10 0.2 1.0 0 -1000
11 0.2 1.0 0 -1000
```

Figure 5.2: Mapping of three AUs to their corresponding FAPs

The Java object model for representation of FACS is presented in Figure 5.3. It provides information about all existing Action Units (AUs), and a container for configurations of values of AUs. `FACS` is static, and when this object is asked to give a list of AUs for the first time, it reads a plain text file that contains all fields of all existing AUs, constructs the necessary `ActionUnit`-objects, stores them in a map and returns it. When an object of the type `FACSConfiguration` is instantiated, an array with twice the number of defined action units is created. This is done because many AUs are asymmetric and have a left and right of the face they describe.

On top of this model, the actual class that the conversion does, is built: `FACSConverter`. The user interface uses `FACSConverter` to update `MPEGConfiguration` when the users drags a slider and hands it over to Xface and FaceEditor. See Figure 5.4 for a screenshot of the main window.

With this prototype built, some small optimizations in the mapping were made. In some cases, it was more realistic to have a certain FAP work on another range of the AU scale than the default 0.0-1.0 range. Or to have other FAPs incorporated in the mapping as well. Things that could not be seen as easily with the MPEG4-direct control utility.

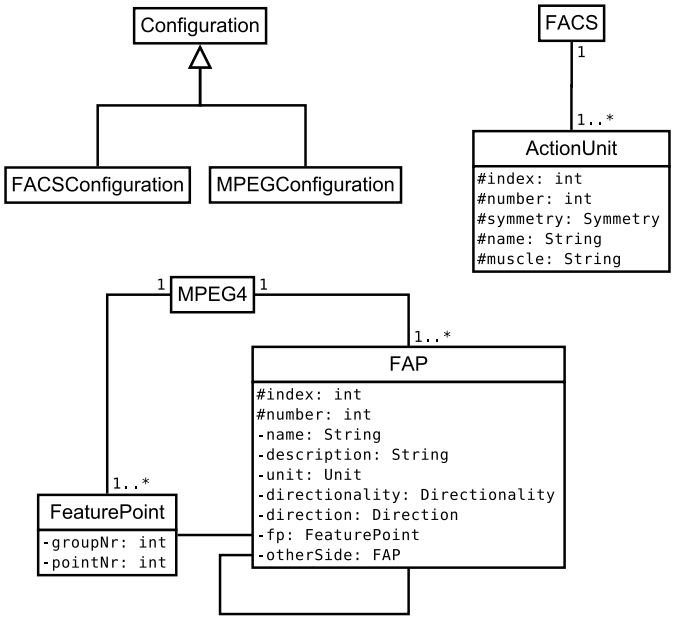


Figure 5.3: Class diagram of the FACS model.

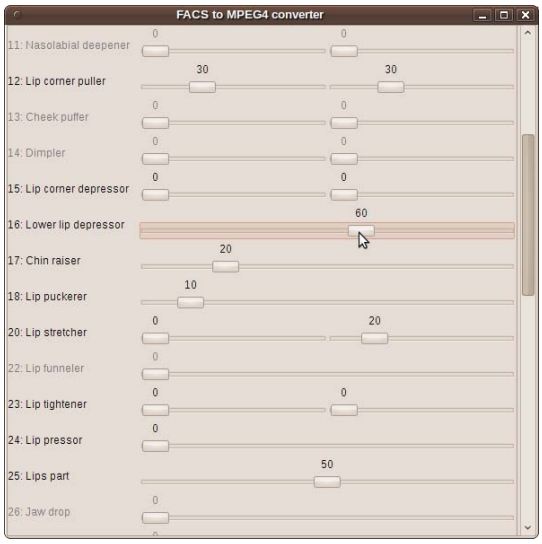


Figure 5.4: Screenshot of the FACS to MPEG4-FA converter.

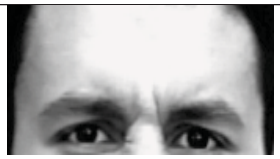
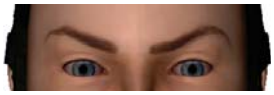
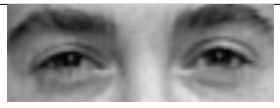
5.3 Evaluation

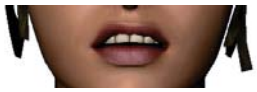
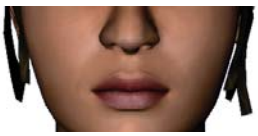
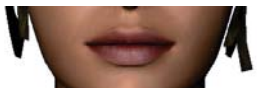
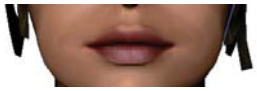
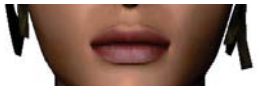
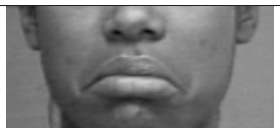
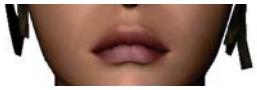
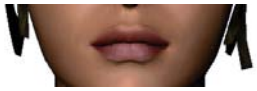
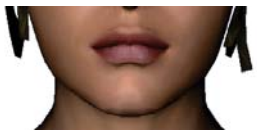
Mapping of AUs to FAPs was done based on Zhang et al. [27] and a set of photographs visually showing the impact of the activation of an AU [17]. This was fine-tuned with the

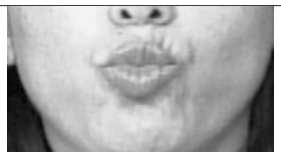
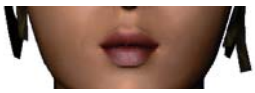
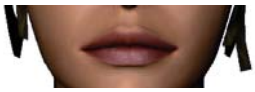
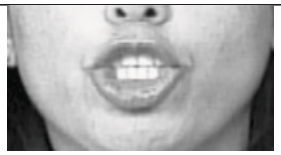
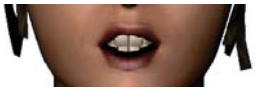
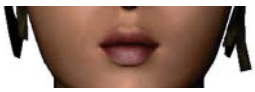
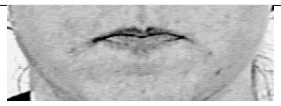
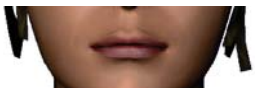
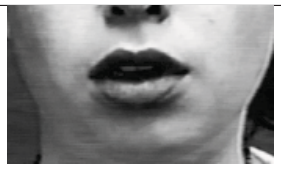
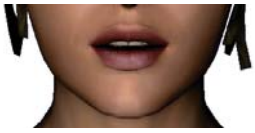
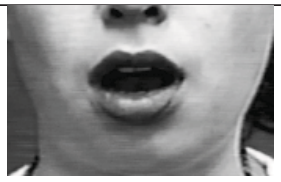
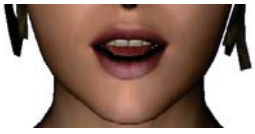
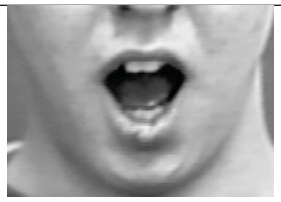
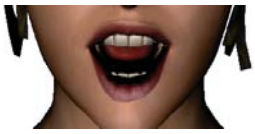
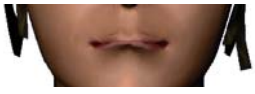
use of Xface and FaceEditor. There are two problems with this:

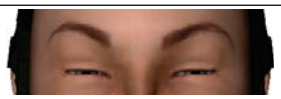
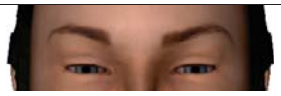
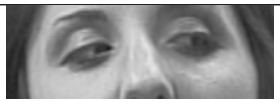
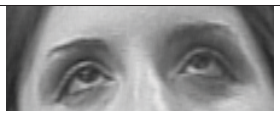
- AUs are not quantified as they only describe that a movement in the face is present and not to what extent. With this converter, AUs can gradually be activated, but FACS does not describe intermediate levels of activation so we're stuck with a simple linear scale.
- MPEG-4 does not have limits on the values for FAPs, so determination of maximum values must be established by hand by looking for which values an MPEG-4 FA face is beginning to look unrealistic which is rather subjective.

See the table below for a comparison between the photographs [17] and screenshots of FaceEditor. Note that for opening the mouth, AUs 26 and 27 are more intensive versions of AU 25. The same holds for closing the eyes: AUs 42 and 43 in respect to AU 41.

Action Unit	CMU	FaceEditor
1: Inner brow raiser		
2: Outer Brow raiser		
4: Brow lowerer		
5: Upper lid raiser		
6: Cheek raiser		
7: Lid tightner		

Action Unit	CMU	FaceEditor
9: Nose wrinkler		
10: Upper lip raiser		
11: Nasolabial deepener		
12: Lip corner puller		
13: Cheek puffer		
14: Dimpler		
15: Lip corner depressor		
16: Lower lip depressor		
17: Chin raiser		

Action Unit	CMU	FaceEditor
18: Lip puckerer		
20: Lip stretcher		
22: Lip funneler		
23: Lip tightner		
24: Lip pressor		
25: Lips part		
26: Jaw drop		
27: Mouth stretch		
28: Lip suck		

Action Unit	CMU	FaceEditor
41: Lid droop		
42: Slit		
43: Eyes closed		
44: Squint		
45: Blink		
46: Wink		
61: Eyes turn left		
62: Eyes turn right		
63: Eyes up		
64: Eyes down		

Eyebrow and eyelid operations pose no problem. There are some slight problems with the lips, especially with AUs 24 (lip pressor) and 28 (lip suck). AU 9 (nose wrinkler) is hardly reproducible since there are no movable MPEG-4 feature points at the base of the nose (see Figure 4.2). But generally speaking, expression produced by FaceEditor are reasonably realistic.

Chapter 6

Conversion from emotion

A way to represent expressions is to describe emotions that led to them. This chapter describes the automatic and real-time translation from emotion to MPEG-4 FA using Plutchik's emotion wheel [21], Whissel's study on activation of emotion terms [25] and is based on previous work by Raouzaiou et al. [22].

6.1 Plutchik's emotion wheel

A fundamental asset to the procedure for conversion of emotions in MPEG-4 FA configuration as proposed by Raouzaiou et al. [22], is Plutchik's emotion wheel, see Figure 6.1. Although it is also displayed as a cone in the image, we only use the flattened version. Plutchik created this wheel from his observation that emotion terms in the space defined by dimensions such as Whissel's are unevenly distributed and tent to form an approximately circular pattern.

Picking an emotion is as easy as picking a point in this two dimension circular space. The emotion picked depends on the distance from the center point and the angle. The greater the distance, the less intense the emotion and vice versa. There are eight basic emotions and eight advanced emotions that are placed in between the basic emotions. Each of the basic and advanced emotions have an opposite although some of these are not that straightforward. See Table 6.1 for a list of emotions in this space.

6.2 Procedure

The primary expressions are the six universally recognizable expressions as proposed by Ekman et al.[9]. For the intermediate expressions, Plutchik's emotion wheel is used.

Basic emotion	Basic opposite	Advanced emotion	Composed of...	Advanced opposite
trust	disgust	submission	trust & fear	contempt
fear	anger	awe	fear & surprise	aggressiveness
surprise	anticipation	disapproval	surprise & sadness	optimism
sadness	joy	remorse	sadness & disgust	love
disgust	trust	contempt	disgust & anger	submission
anger	fear	aggressiveness	anger & anticipation	awe
anticipation	surprise	optimism	anticipation & joy	disapproval
joy	sadness	love	joy & trust	remorse

Table 6.1: List of basic and advanced emotions in Plutchik's emotion space.

Raouzaiou et al. [22] quantified FAPs for all archetypal emotions with input from psychological studies and experimental data provided by classic databases as Ekman's and MediaLab's. For each archetypal emotion, several profiles (valid instances of the archetypal emotion) have been made. Each profile contains a set of FAPs and their corresponding range of variation. One of the profiles for fear is given in Figure 6.2. Ranges of variation are created using the mean and standard deviation coming from the experimental data and provide room for fuzziness and mild adjustments to reduce "robot-like" expressions. In the current implementation, ranges are simply reduced to their midpoint and length.

When calculating FAPs for intermediate expressions (the visual portion of emotion), Raouzaiou et al. considered two different cases:

1. emotions that are similar, in nature, to an archetypal one; for example they may differ only in the intensity of muscle actions;
2. emotions that cannot be considered as related to any of the archetypal ones.

For calculating FAPs for intermediate expressions, first have a look at Figure 6.3. P_A is the location of the archetypal emotion (the activation parameter for the emotion word *surprise* is 6.5) and P_I is the location for which we want to calculate a new profile. Calculation is done as follows (based on Raouzaiou et al. [22]):

Let P_i be a profile of emotion i and $X_{i,j}$ be the range of variation of FAP F_j involved in P_i. If A, I are emotions belonging to the same universal emotion category, A being the archetypal, and I the intermediate one, then the following rules are applied:

1. P_A and P_I employ the same FAPs.
2. a_A and a_I are the values of the *activation* parameter for emotion words A and I obtained from Whissel's study [25].
3. The range of variation $X_{I,j}$ is computed by $X_{I,j} = (a_I/a_A)X_{A,j}$.

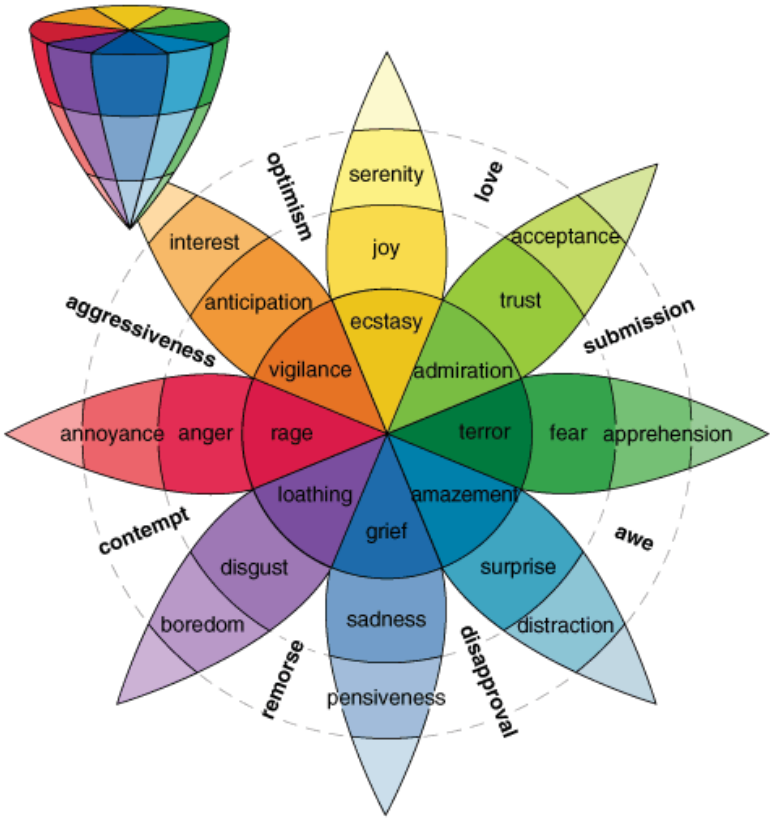


Figure 6.1: Plutchik's model of emotion. It describes relations among emotion concepts.

In other words: the range is calculated using the ratio between the activation of P_A and P_I. Let us take FAP 3 (open_jaw) of one of the surprise-profiles as example. It has range $[569, 1201]$ and P_A has activation 6.5 and P_I has activation 4.0 so the new range is

$$[569 * (4.0/6.5), 1201 * (4.0/6.5)] = [350, 739] \tag{6.1}$$

This is then done for all FAPs contained in the selected profile.

Calculating FAPs for intermediate expressions that lie in between two archetypal emotions (in between on the wheel of Plutchik [21]) is slightly more complicated. The following is the formal description (also based on from Raouzaiou et al.):

Let P_{A_1} be a profile of emotion A_1 and P_{A_2} a profile of emotion A_2, then the following rules are applied so as to create a profile P_I for the intermediate emotion I.

1. P_I includes FAPs that are involved either in P_{A_1} or P_{A_2}.

$F_3 \in [400, 560], F_5 \in [-240, -160], F_{19} \in [-630, -570], F_{20} \in [-630, -570], F_{21} \in [-630, -570], F_{22} \in [-630, -570], F_{31} \in [460, 540], F_{32} \in [460, 540], F_{37} \in [60, 140], F_{38} \in [60, 140]$

Figure 6.2: A profile for fear. See Raouzaiou et al. [22] for a complete listing of all profiles for all archetypal emotions.

2. a_{A_1}, a_{A_2} and a_I are the values of the *activation* parameter for emotion words A_1, A_2 and I obtained from Whissel's study [25].

3. ω_{A_1}, ω_{A_2} and ω_I, $\omega_{A_1} \leq \omega_I \leq \omega_{A_2}$ are the *angular* parameters for emotion words A_1, A_2 and I, obtained from Plutchik's study [21].

4. If F_j is a FAP involved in both P_{A_1} and P_{A_2} with the same sign (direction of movement), then the range of variation $X_{I,j}$ is computed as a weighted translation of $X_{A_1,j}$ and $X_{A_2,j}$ (where $X_{A_1,j}$ and $X_{A_2,j}$ are the ranges of variation of FAP F_j involved in P_{A_1} and P_{A_2}, resp.) in the following way:

   (a) we compute the translated range of variations

   $$t(X_{A_1,j}) = \frac{a_I}{a_{A_1}} X_{A_1,j}, \qquad t(X_{A_2,j}) = \frac{a_I}{a_{A_2}} X_{A_2,j} \tag{6.2}$$

   of $X_{A_1,j}$ and $X_{A_2,j}$,

   (b) we compute the center and length $c_{A_1,j}$ and $s_{A_1,j}$ of $t(X_{A_1,j})$ and $c_{A_2,j}$ and $s_{A_2,j}$ of $t(X_{A_2,j})$,

   (c) the length of $X_{I,j}$ is

   $$s_{I,j} = \frac{\omega_I - \omega_{A_1}}{\omega_{A_2} - \omega_{A_1}} s_{A_1,j} + \frac{\omega_{A_2} - \omega_I}{\omega_{A_2} - \omega_{A_1}} s_{A_2,j} \tag{6.3}$$

   and its midpoint is

   $$c_{I,j} = \frac{\omega_I - \omega_{A_1}}{\omega_{A_2} - \omega_{A_1}} c_{A_1,j} + \frac{\omega_{A_2} - \omega_I}{\omega_{A_2} - \omega_{A_1}} c_{A_2,j}. \tag{6.4}$$

5. If the F_j is involved in both P_{A_1} and P_{A_2} but with a contradictory sign (opposite direction of movement), then the range of variation $X_{I,j}$ is computed by

   $$X_{I,j} = \frac{a_I}{a_{A_1}} X_{A_1,j} \cap \frac{a_I}{a_{A_2}} X_{A_2,j}. \tag{6.5}$$

   In case where $X_{I,j}$ is eliminated (which is the most likely situation), F_j is excluded from the profile.

6. If the F_j is involved only in one of P_{A_1} and P_{A_2}, then the range of variation $X_{I,j}$ will be averaged with the neutral face, that is, $X_{I,j} = (a_I / (2 * a_{A_1})) X_{A_1,j}$ or $X_{I,j} = (a_I / (2 * a_{A_2})) X_{A_2,j}$.

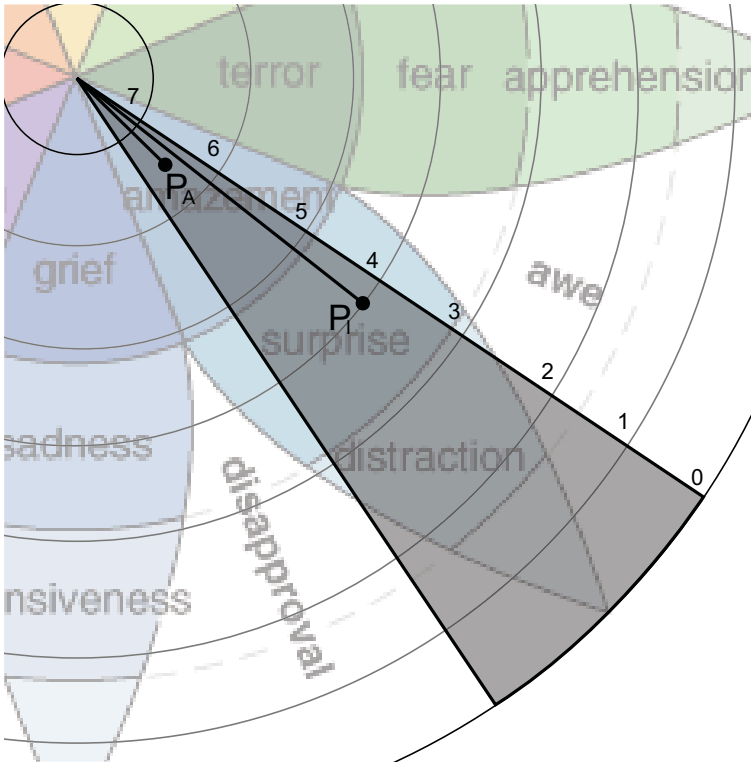


Figure 6.3: A situation in which a new profile is calculated within the same archetypal emotion.

When estimating ranges of variation of FAPs for profiles for emotions that do not clearly belong to a universal category, *activation* and the *angular* measures are used. In the approach of Raouzaiou et al. [22], FAPs that are common in both emotions are retained during synthesis and FAPs that only exist in one of the emotions are averaged with the neutral face. FAPs with contradicting intensities are canceled out.

Have a look at Figure 6.4 which will be held as example. a_{A_1} is the first archetypal emotion, anger. Its angle is 225° and its activation (according to Whissel) is 2.4. The second archetypal emotion is a_{A_2}, joy. Its angle is 315° and its activation is 5.4. We want to construct a profile for a_I. Its angle is 261° and its activation is 3.8.

Now lets have a look at the procedure for FAP 3 (open_jaw). This is an easy one because it is only present in the profile for joy. Its range is $[195, 205]$. This FAP will be included in a_I but averaged with the neutral face:

$$[(3.8/(2 * 5.4)) * 195 = 67, (3.8/(2 * 5.4)) * 205 = 72]. \tag{6.6}$$

FAP 20 (close_t_r_eyelid) is included in both profiles. For the second profile of anger it is $[-335, -205]$ and for the fourth profile of joy $[-426, -302]$. First, we calculate translated

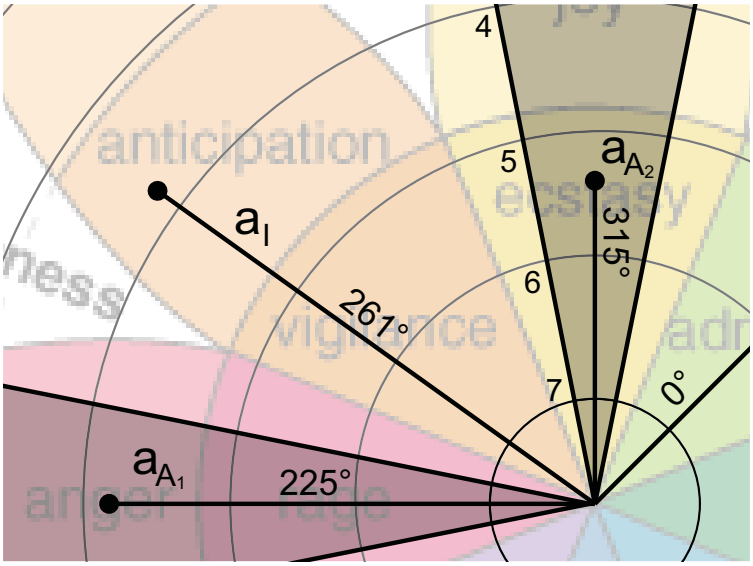


Figure 6.4: A situation in which a new profile (a_I) is calculated which lies in between two different archetypal emotions.

ranges of variation. For anger:

$$[(3.8/4.2) * -335 = -303, (3.8/4.2) * -205 = -185]. \tag{6.7}$$

Center = -260, length = 150. For joy:

$$[(3.8/5.4) * -426 = -300, (3.8/5.4) * -302 = -213]. \tag{6.8}$$

Center = -256.5, length = 87.

The length of the new range:

$$\frac{\omega_I - \omega_{A_1}}{\omega_{A_2} - \omega_{A_1}} = \frac{261 - 225}{315 - 225} = 0.4, \tag{6.9}$$

$$\frac{\omega_{A_2} - \omega_I}{\omega_{A_2} - \omega_{A_1}} = \frac{315 - 261}{315 - 225} = 0.6, \tag{6.10}$$

$$s_{I,j}^{(m)} = 0.4 * 150 + 0.6 * 87 = 112.2, \tag{6.11}$$

and the center:

$$c_{I,j}^{(m)} = 0.4 * -260 + 0.6 * -256.5 = -257.9, \tag{6.12}$$

so the range of FAP 20 in the new intermediate profile is $[-314, -202]$.

6.3 Our emotion conversion implementation

The prototype consists of:

- a file format for representation of profiles of all archetypal emotions;
- a Java model to represent FAPs
- a Java application that reads the data file, supplies the user with a GUI with which he can choose different profiles and positions on the circulair model of emotion of Plutchik and interfaces with Xface and FaceEditor.

The file format is a, just as the mapping file for the FACS conversion, simple and plain. It is text based and each significant line can either be a profile label or a profile. Each profile label precedes exactly one profile. A profile label is build up from the archetypal emotion it represents, a space, and a sequence number which is unique within the scope of an emotion. A profile is a comma seperated list of FAP ranges and each FAP consists of a reference to a FAP and a lower and upper boundary.

See Figure 6.5 for a few lines from a file in the format described above. The three lines with profiles have been shortened to improve readability. We can see in this fragment that, for example, for the 10th profile for Fear, the range for FAP 5 (raise_b_midlip) starts at 307 and ends at 399. The angle and activation of the archetypal emotions themselves are set in Java code.

```
Fear 9
F3 [400, 560], F5 [307, 399], ..., F36 [460, 540]
Surprise 0
F3 [569, 1201], F5 [340, 746], ..., F54 [-121,-43]
Surprise 1
F3 [1150, 1252], F5 [-792,-700], ..., F54 [-141,-101]
```

Figure 6.5: Three profiles for two different emotions

The real work is done in a class called `EmotionConverter`, but some classes represented in Figure 6.6 are used in the process. These classes were also used in the FACS conversion prototype.

The user interface uses `EmotionConverter` to update `MPEGConfiguration` when a user clicks somewhere in the emotion space and hands it over to Xface and FaceEditor. See Figure 6.7 for a screenshot of the main window.

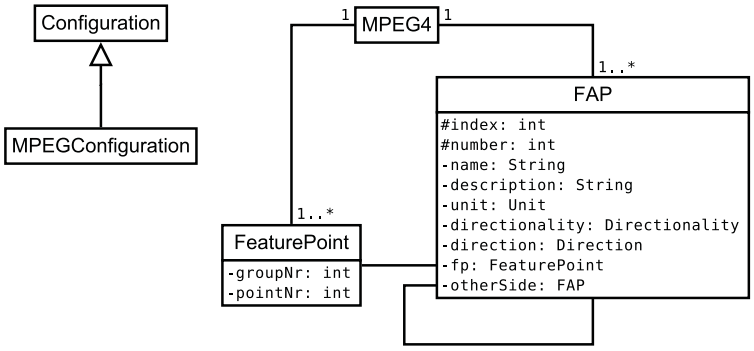


Figure 6.6: Class diagram of the auxiliary classes.

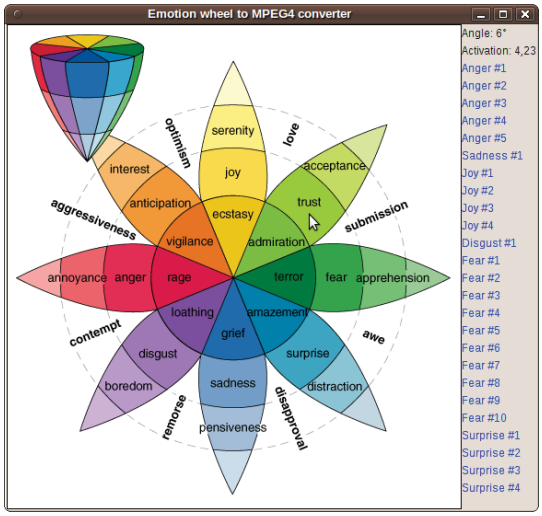


Figure 6.7: Screenshot of the emotion to MPEG4-FA converter.

6.4 Evaluation

With the prototype built, we did a preliminary assessment of the quality of the produced faces and the profiles supplied by Raouzaiou et al. It was disappointing and we investigated it a bit further in the next few paragraphs. See §7.3 for a discussion of the method used by Raouzaiou et al. for aquisition of the data.

For evaluation purposes, they use a face model developed in the context of the European project ACTS MoMuSys [4]. It was freely available but it has evolved in a reference implementation and is now part of the MPEG-4 standard itself so we can not use it. However, Raouzaiou et al. included screenshots in their work. We will not only compare these screenshots with the performance of FaceEditor, but also with the three other faces

we have used earlier in evaluation of FaceEditor itself, see §4.4.1.

Raouzaiou et al. have displayed screenshots of MoMuSys for three different profiles for anger, two for surprise and one for joy. To illustrate the method of creation of new profiles, they also included images for the emotion terms *afraid*, *terrified* and *worried*, *afraid*, *guilty* and *sad*. Because we don't know which profiles are used for animation of the face used in MoMuSys - except for sadness and joy because they each have only one profile - we cannot reconstruct the used FAP configurations exactly. In these cases, the profile that looks best in FaceEditor will be selected.

See Figure 6.8 for the comparison. This shows that especially for anger and joy, the supplied profiles result in unrealistic faces.

Based on this, we chose to hand-craft each archetypal emotion by ourself to see if we can produce some better profiles. As guide for what FAPs we need to activate for each archetypal emotion, we still used the vocabularies of the aforementioned work. See Figure 6.9 for a visual display of both the best profiles supplied by Raouzaiou et al. and the hand-crafted versions. Problems clearly arise for anger and joy.

To show that the new profiles perform well for all other faces used in evaluation earlier, they will be put side to side. See Figure 6.10. Only the profile created for anger is slightly of course for all faces but FaceEditor.

To further explore and evaluate the emotion space and the conversion method, we will simply show faces that correspond to points on a circular grid that is placed in the emotion space. Note that the angle 0° is in between joy and fear (trust) and that it is clockwise. See the following table.

	Activation				Emotion
Angle	1	3	5	7	
0.0					Trust
22.5					Submission
45.0					Fear

	Activation				Emotion
Angle	1	3	5	7	
67.5					Awe
90.0					Surprise
112.5					Disapproval
135.0					Sadness
157.5					Remorse
180.0					Disgust
202.5					Contempt
225.0					Anger
247.5					Aggressiveness

	Activation				Emotion
Angle	1	3	5	7	
270.0					Anticipation
292.5					Optimism
315.0					Joy
337.5					Love
360.0					Trust

The process of conversion or translation of a certain emotion in a FAP configuration has been implemented with help from Raouzaiou et al. The method seems to work well. The data was less usable, but new profiles could easily be created by hand.

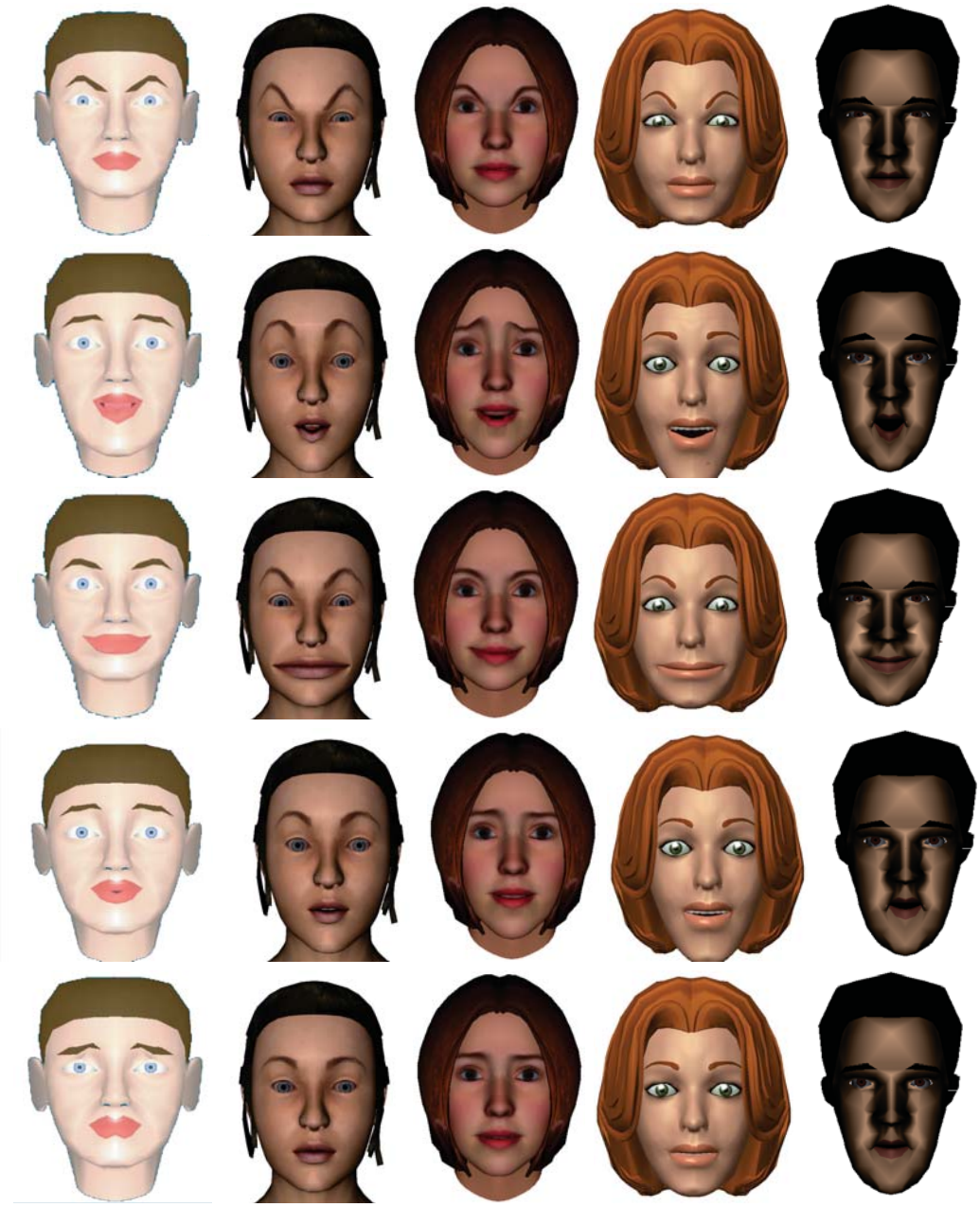


Figure 6.8: Screenshots of MuMoSys, FaceEditor, Xface, Greta and Miraface. The screenshots of MuMoSys are taken from the paper by Raouzaiou et al., the other screenshots are made based on reconstructed profiles that represent the same FAP configuration as the one the MuMoSys screenshots were taken from. From top to bottom: anger, surprise, joy, terrified and sad.

Figure 6.9: Screenshots of FaceEditor for the six archetypal emotions. The top row contains the best performing profiles coming from Raouzaiou et al., the bottom row contains the hand crafted ones. From left to right: joy, sadness, anger, fear, disgust and surprise.

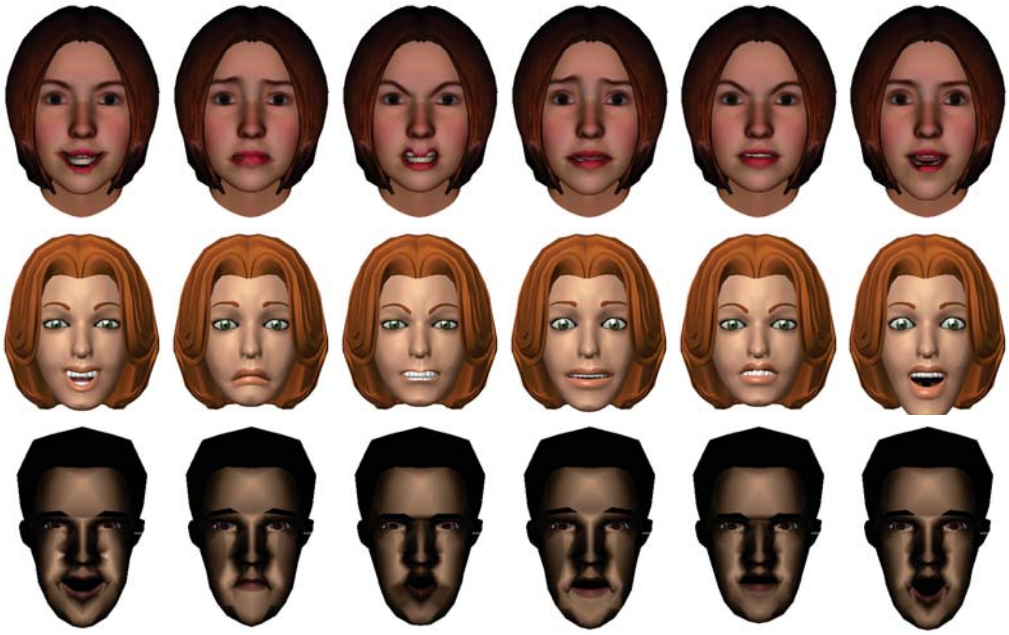


Figure 6.10: The new archetypal profiles displayed in Xface, Greta and Miraface. From left to right: joy, sadness, anger, fear, disgust and surprise.

Chapter 7

Discussion

In this chapter, we will talk about some of the weaknesses and strengths of some of the methods that have been developed or reused.

7.1 MPEG-4 Facial Animation

The MPEG-4 Facial Animation (FA) standard is clear and sound for the part it describes. It describes FPs (Feature Points) on the face and the direction and distance they should travel in relation to FAPs (Facial Action Parameters). The mapping between this displaced FP and how the vertices should be displaced is left open for the implementation and this is done because it is not feasible to globally specify this mapping for all different facial vertex topologies.

This leaves us with the following question: how to create and evaluate a certain mapping created for a certain face? In contrast with FACS, a lot of individual FAPs cannot be recproduced individually by a human face. So this mapping must be created solely by means of the tools created to define these mappings for faces and the one who sets the parameters for a particular face. The input for this process is the description given by the standard and the personal interpretation of the animator. This description is short, but nevertheless the total range of mappings is limited because there are only a few descriptions of FAPs that allow for multiple interpretations.

An example of a FAP description that is not entirely clear is that of the midpoint lip displacements. The position of the FP is known, so is the distance it travels, but it is unknown what the width of the influence area must be exactly. The impact of a different interpretation in this case can however be easily be limited by another layer on top of the sum of all vertex displacements (vectors) which checks that if a vertex has vector in the same direction from more than one FAP, one of those vectors is canceled. This check comes

in effect when two influence areas overlap each other and - when activated simultaneously - create a w-like shape. See Figure 7.1 for a visual example. We solved this by easing, but it must be set in cooperation with the size of the influence sphere and this is still a point in which two faces can be configured differently.

Figure 7.1: From left to right: neutral upper lips, the way it should be when the three upper lips points are moved up and the way it looks when the middle one has a influence sphere that is too large.

Evaluation of a specific implementation of MPEG-4 FA can not be done by means of a quantitative research. Because of the unnatural facial expressions (because of activation of only individual FAPs), a reference to what is good must be given. We have found some sort of reference implementation by looking at other faces that implemented MPEG-4 FA but when asking subjects which face is producing the best expression for any of the FAPs, they inevitably will at least take other irrelevant information in consideration, if they are not led by them in the first place. There are numerous factors that would distract subjects, such as how the face itself looks, hair, textures, geometry, etcetera. The only sensible alternative was to assess all FAPs on all available faces one by one and give scores that evaluated the displacements individually.

Our way of implementation and evaluation of this implementation gives us enough reason to believe the implementation is usable in an environment where expressions and animations based on FAPs are interchangeable with other faces displayed by the same and other implementations of MPEG-4 FA.

7.2 Conversion from FACS

FACS is one of the higher level control mechanisms. It involves a translation from configurations of AU (Action Unit) values to configurations of FAP values. FACS claims to have indexed all individually discriminable expressions that a human face can show. And since each and every movement in the face also can be represented using FAPs, a translation should be possible.

This has proven to be true to a large extent, but not completely. The largest problem is AU 9, the nose wrinkler. It involves movement of two points at the base of the nose. These points correspond to points in MPEG-4 FA, namely 9.4 and 9.5 (see Figure 4.2) but the problem is that no FAP has effect on their position so they cannot be moved. Points

9.2 and 9.1 (respectively the points ahead of points 9.4 and 9.5) are movable by FAPs 61 (stretch_l_nose) and 62 (stretch_r_nose), but only on a horizontal line through these points from left to right. A way of giving a face a nose wrinkle using MPEG-4 FA is to map a expression (FAP 2) to a custom deformation target, but this is less portable.

Input for creation of the mapping was a set of images visually showing the impact of the activation of an AU, courtesy of the Carnegie Mellon University School of Computer Science [17]. FAPs were set per AU to mimic the expression shown in the real life photograph.

Since all FACS can be reproducible by the human face, evaluation of this conversion meant matching up actual images of FACS performances by a human face with the actual results produced by FaceEditor. In addition or alternative to this, an official FACS coder could have been hired that could have validated that each and every face shows activation of the right AU and maybe even could have assessed the quality, but they are expensive. And since we have real life imagery for all individual AUs, we chose to simply show the imagery side by side.

7.3 Conversion from emotion

The other higher level control that has been described earlier, is the one that converts a position in a two dimensional emotion space into a configuration of FAP values. It relies heavily on the work of Raouzaiou et al. [22] in two ways. First, we implemented their method for synthesis of new FAP profiles (a FAP configuration with a range of variation for each FAP) based on profiles for archetypal emotions. Secondly, these base profiles for the archetypal emotions were being filled in by the data that was the result of another process.

This other process consisted of a few steps. Raouzaiou et al. [22] wrote: "we translate facial muscle movements – describing expressions through muscle actions – into FAPs and create a vocabulary of FAPs for each archetypal expression. [They] are also experimentally verified through analysis of prototype datasets." Furthermore, ranges of variation (to be able to create the profiles) were estimated for each archetypal profile by "analyzing real images and video sequences as well as by animating synthesized examples".

The method for synthesis of new FAP profiles performed well, but the archetypal profiles themselves were not producing realistic faces on our side in FaceEditor at all. Because of the claims made by Raouzaiou et al. and the imagery showed in their paper, and to rule out that FaceEditor was interpreting or showing these profiles wrong, this issue has been investigated further in §6.4. It shows that when visualizing these profiles, unrealistic results appear on more faces. The fact that MuMoSys (the face used by Raouzaiou et al.) showed viable results could have been caused by the relative low number of polygons. Just like Miraface, it is a old face model which can lead to extremes being flattened out. This is supported by the fact that all displays of Miraface in Figure 6.8 look like MuMoSys the

most.

In general, the data collection method of Raouzaiou et al. proved to be usable for MuMoSys for a limited number of easily detectable feature points. But variance in the dataset they created is large and other errors may have been masked by the fact that MuMoSys is rather low polygon in terms of todays face models.

7.4 Combination of higher level controls

It is possible to combine two or more higher level steering elements. For example, one could deform the face using a position on the two dimensional emotion space and alter this with the FACS controller. There is a limitation in that one FAP can only be set once. When it is set a second time, the old value is overwritten. And because in general, higher level controls with a few input parameters affect a lot of FAPs at the same time, it is advisable to always set the expression using the method that has the fewest input parameters first and then refine it with methods with more input parameters. See Figure 7.2 for what can be accomplished for example.

Figure 7.2: Combination of a wink set using the FACS converter and serenity using the emotion converter.

7.5 Conclusion

In this chapter, we have spoken about weaknesses and strengths of MPEG-4 FA, conversion from FACS and and the conversion from emotions. Also, the way in which these parts are evaluated is discussed.

Chapter 8

Conclusion and future work

8.1 Conclusion

To conclude this thesis, we will revisit the objectives that are given in the introduction of this thesis en describe for each objective to what extent and how is has been fulfilled. These objectives were to design and implement a set of tools that:

1. assist in creating facial expressions by providing several high level steering instruments that can be driven by a limited number of parameters;
2. provide a good trade of between number of control parameters and range of expression;
3. actually apply micro adjustments to virtual faces in such a way that these faces are interchangeable with other faces without changing too much of the expression;
4. interface with Behavior Markup Language (BML) and
5. work in real-time with respect to three dimensional rendering on the screen.

Point 1: two high level steering instruments have been designed and implemented. These respectively translate FACS and emotion into the low level expression representation MPEG-4 FA. The translation from FACS performs well, only slight problems occur in translation of FACS' Action Unit 9 (nose wrinkler) because there are no corresponding MPEG-4 FA feature points at the base of the nose. Generally speaking, expression that come out of this translation are reasonably realistic.

Regarding the translation from emotion, re-using data from Raouzaiou et al. [22] needed for this translation, resulted in unrealistically looking faces. We hand-crafted our own data and with this, the method itself proved to be usable.

Point 2: the two provided high level steering instruments are different from each other in the sense that the translation from FACS has a relatively high number of input parameters and the translation from emotion only two. With this translation from FACS, almost every facial expression can be built, but with the translation from emotion, a complete expression can be made by only choosing an emotion and an intensity. Furthermore, when combining these two high level steering instruments, the emotion instrument can be used to set a global face which can then be further altered by using the FACS instrument.

Point 3: the software package created enables to create parameter files for faces that need to be expression-enabled. Some work still needs to be done before another face can be used but a graphical user interface is provided to make the mandatory labor of setting parameters less intensive.

Point 4: a BML reader has been designed and developed and this is the first step in BML as container for high and low level expression representations. It can contain FACS action units, timing constraints and it may also have FAP values or emotion space coordinates in it using extensions. Furthermore, this implementation can be used for all other things BML is capable of representing, such as body animation.

Point 5: all operations used in the context of three dimensional rendering are cheap; the high level steering instruments and the application of micro adjustments work in real-time.

8.2 Future work

Future work related to the work presented in this thesis is represented in the following points.

- Since all implemented facial expressions do not yet take pre-modelled visemes or morph targets in general into account, actions have to be taken in order to be able to create a embodied *conversational* agent. There are several paths that can be traveled in order to achieve this. Mixing for example. Care must be taken with respect to priorities and conflicts between the two sources of facial deformation. Another possibility is to recreate visemes as FAP-configurations and attack this problem at FAP-level.
- Continuing on the course of the previous point, the same problems arises when heavily combining higher level control methods. The controls are not aware of the presence of other control methods. One might also think in the direction of a complete hierarchy of control methods, all working together. Not only on the same level but also in the branches of the tree.
- In this work, we only looked at static expression. We believe that animation of expressions or facial animation can improve realism and recognition of facial expres-

sions, so future research regarding this subject would be to find out how expressions must be turned into facial animation in such a way that it improves realism and recognition of expressions.

- Not relevant to all face models, but the face model used for this thesis incorporated eyelashes. For the sake of simplicity, these were hidden because it might take a considerate amount of time to automatically align deformation of these objects with the actual movements of the eyelids. This is something to consider for production quality environments. Other auxiliary objects are related to this, such as animation of hair or dilation of the eye pupils.

- Incorporation of wrinkles can improve realism. In cases where the skin would normally wrinkle (such as on the forehead when frowning), the virtual skin should do the same. Several attempts to do this have been made, see Stefano Pasquariello and Catherine Pelachaud [20] or Wu, Kalra, Moccozet and Magnenat-Thalmann[26] for example.

- The emotion conversion model uses the activation parameter from Whissel's study [25] for emotion terms in cooperation with Plutchik's emotion wheel [21]. But it also incorporates other terms or factors, evaluation being the second most important after activation. There is not a 1:1 relationship between this evaluation parameter and angles from Plutchik's emotion wheel. It might be worthwhile to investigate this extra parameter and maybe extend the emotion conversion model with it. It would also be worthwhile to look at completely different emotion models such as the widely used valence-arousal representation.

- Researches have made attempts for automatic feature point placement. Not limited to the field of analysis of real life imagery, but also for synthesis purposes. Research could be done to investigate whether automatic placement of MPEG-4 FA feature points on the face is feasible. An advantage above analysis of real life image data is that the algorithm has access to the vertex topology. One could imagine that corners of the mouth or eyes should not be hard to find.

- When vertices are displaced, their movement is bound to the direction in which the feature point is moving for a particular FAP. Research could be done to find out if realism can be improved by other ways of defining new vertex positions not only by their distance to the feature point and its direction. More skin like deformations for example.

- When vertices are displaced, the size of the displacements are based on a function of the distance to the feature point with respect to the size of the influence sphere. This function is now limited to exponential functions and smoothing using Bézier curves but other functions such as raised cosine could lead to further improvement.

- The whole idea of the influence sphere, easing and masking is to assign weights to vertices. Numerous other procedures can be followed here, such as vertex weight

painting or defining regions on the face. This can also be combined with work regarding the directions vertices should travel. And all of this could be defined globally or on a per-FAP or per-feature point basis.

Appendix A

FACS Action Units

AU	Description	Facial muscle
1	Inner Brow Raiser	Frontalis, pars medialis
2	Outer Brow Raiser	Frontalis, pars lateralis
4	Brow Lowerer	Corrugator supercilii, Depressor supercilii
5	Upper Lid Raiser	Levator palpebrae superioris
6	Cheek Raiser	Orbicularis oculi, pars orbitalis
7	Lid Tightener	Orbicularis oculi, pars palpebralis
9	Nose Wrinkler	Levator labii superioris alaeque nasi
10	Upper Lip Raiser	Levator labii superioris
11	Nasolabial Deepener	Zygomaticus minor
12	Lip Corner Puller	Zygomaticus major
13	Cheek Puffer	Levator anguli oris (a.k.a. Caninus)
14	Dimpler	Buccinator
15	Lip Corner Depressor	Depressor anguli oris (a.k.a. Triangularis)
16	Lower Lip Depressor	Depressor labii inferioris
17	Chin Raiser	Mentalis
18	Lip Puckerer	Incisivii labii superioris and Incisivii labii inferioris
20	Lip stretcher	Risorius with platysma
22	Lip Funneler	Orbicularis oris
23	Lip Tightener	Orbicularis oris

AU	Description	Facial muscle
24	Lip Pressor	Orbicularis oris
25	Lips part[a]	Depressor labii inferioris or relaxation of Mentalis, or Orbicularis oris
26	Jaw Drop	Masseter, relaxed Temporalis and internal Pterygoid
27	Mouth Stretch	Pterygoids, Digastric
28	Lip Suck	Orbicularis oris
41	Lid droop[b]	Relaxation of Levator palpebrae superioris
42	Slit	Orbicularis oculi
43	Eyes Closed	Relaxation of Levator palpebrae superioris; Orbicularis oculi, pars palpebralis
44	Squint	Orbicularis oculi, pars palpebralis
45	Blink	Relaxation of Levator palpebrae superioris; Orbicularis oculi, pars palpebralis
46	Wink	Relaxation of Levator palpebrae superioris; Orbicularis oculi, pars palpebralis
51	Head turn left	
52	Head turn right	
53	Head up	
54	Head down	
55	Head tilt left	
56	Head tilt right	
57	Head forward	
58	Head back	
61	Eyes turn left	
62	Eyes turn right	
63	Eyes up	
64	Eyes down	

[a]This currently also embeds AU's 26 and 27 by intensity.
[b]This currently also embeds AU's 42 and 43 by intensity.

Table A.1: The Action Units defined in FACS

Appendix B

MPEG-4 FA Facial Action Parameters

FAP	Name	Description	Unit	Uni- / bidirectional	Motion	Feature point
1	viseme	Set of values determining the mixture of two visemes for this frame (e.g. pbm, fv, th)				
2	expression	A set of values determining the mixture of two facial expression				
3	open_jaw	Vertical jaw displacement (does not affect mouth opening)	MNS	U	down	2.1
4	lower_t_midlip	Vertical top middle inner lip displacement	MNS	B	down	2.2
5	raise_b_midlip	Vertical bottom middle inner lip displacement	MNS	B	up	2.3

FAP	Name	Description	Unit	Uni- / bidirectional	Motion	Feature point
6	stretch_l_cornerlip	Horizontal displacement of left inner lip corner	MW	B	left	2.4
7	stretch_r_cornerlip	Horizontal displacement of right inner lip corner	MW	B	right	2.5
8	lower_t_lip_lm	Vertical displacement of midpoint between left corner and middle of top inner lip	MNS	B	down	2.6
9	lower_t_lip_rm	Vertical displacement of midpoint between right corner and middle of top inner lip	MNS	B	down	2.7
10	raise_b_lip_lm	Vertical displacement of midpoint between left corner and middle of bottom inner lip	MNS	B	up	2.8
11	raise_b_lip_rm	Vertical displacement of midpoint between right corner and middle of bottom inner lip	MNS	B	up	2.9
12	raise_l_cornerlip	Vertical displacement of left inner lip corner	MNS	B	up	2.4
13	raise_r_cornerlip	Vertical displacement of right inner lip corner	MNS	B	up	2.5
14	thrust_jaw	Depth displacement of jaw	MNS	U	forward	2.1
15	shift_jaw	Side to side displacement of jaw	MW	B	right	2.1
16	push_b_lip	Depth displacement of bottom middle lip	MNS	B	forward	2.3
17	push_t_lip	Depth displacement of top middle lip	MNS	B	forward	2.2

FAP	Name	Description	Unit	Uni- / bidirectional	Motion	Feature point
18	depress_chin	Upward and compressing movement of the chin (like in sadness)	MNS	B	up	2.10
19	close_t_l_eyelid	Vertical displacement of top left eyelid	IRISD	B	down	3.1
20	close_t_r_eyelid	Vertical displacement of top right eyelid	IRISD	B	down	3.1
21	close_b_l_eyelid	Vertical displacement of bottom left eyelid	IRISD	B	up	3.3
22	close_b_r_eyelid	Vertical displacement of bottom right eyelid	IRISD	B	up	3.4
23	yaw_l_eyeball	Horizontal orientation of left eyeball	AU	B	left	3.5
24	yaw_r_eyeball	Horizontal orientation of right eyeball	AU	B	left	3.6
25	pitch_l_eyeball	Vertical orientation of left eyeball	AU	B	down	3.5
26	pitch_r_eyeball	Vertical orientation of right eyeball	AU	B	down	3.6
27	thrust_l_eyeball	Depth displacement of left eyeball	ES	B	forward	3.5
28	thrust_r_eyeball	Depth displacement of right eyeball	ES	B	forward	3.6
29	dilate_l_pupil	Dilation of left pupil	IRISD	B	growing	3.5
30	dilate_r_pupil	Dilation of right pupil	IRISD	B	growing	3.6
31	raise_l_i_eyebrow	Vertical displacement of left inner eyebrow	ENS	B	up	4.1
32	raise_r_i_eyebrow	Vertical displacement of right inner eyebrow	ENS	B	up	4.2
33	raise_l_m_eyebrow	Vertical displacement of left middle eyebrow	ENS	B	up	4.3

FAP	Name	Description	Unit	Uni- / bidirectional	Motion	Feature point
34	raise_r_m_eyebrow	Vertical displacement of right middle eyebrow	ENS	B	up	4.4
35	raise_l_o_eyebrow	Vertical displacement of left outer eyebrow	ENS	B	up	4.5
36	raise_r_o_eyebrow	Vertical displacement of right outer eyebrow	ENS	B	up	4.6
37	squeeze_l_eyebrow	Horizontal displacement of left eyebrow	ES	B	right	4.1
38	squeeze_r_eyebrow	Horizontal displacement of right eyebrow	ES	B	left	4.2
39	puff_l_cheek	Horizontal displacement of left cheeck	ES	B	left	5.1
40	puff_r_cheek	Horizontal displacement of right cheeck	ES	B	right	5.2
41	lift_l_cheek	Vertical displacement of left cheek	ENS	U	up	5.3
42	lift_r_cheek	Vertical displacement of right cheek	ENS	U	up	5.4
43	shift_tongue_tip	Horizontal displacement of tongue tip	MW	B	right	6.1
44	raise_tongue_tip	Vertical displacement of tongue tip	MNS	B	up	6.1
45	thrust_tongue_tip	Depth displacement of tongue tip	MW	B	forward	6.1
46	raise_tongue	Vertical displacement of tongue	MNS	B	up	6.2
47	tongue_roll	Rolling of the tongue into U shape	AU	U	concave upward	6.3 & 6.4

FAP	Name	Description	Unit	Uni- / bidirectional	Motion	Feature point
48	head_pitch	Head pitch angle from top of spine	AU	B	down	7.1
49	head_yaw	Head yaw angle from top of spine	AU	B	left	7.1
50	head_roll	Head roll angle from top of spine	AU	B	right	7.1
51	lower_t_midlip _o	Vertical top middle outer lip displacement	MNS	B	down	8.1
52	raise_b_midlip_o	Vertical bottom middle outer lip displacement	MNS	B	up	8.2
53	stretch_l_cornerlip_o	Horizontal displacement of left outer lip corner	MW	B	left	8.3
54	stretch_r_cornerlip_o	Horizontal displacement of right outer lip corner	MW	B	right	8.4
55	lower_t_lip_lm _o	Vertical displacement of midpoint between left corner and middle of top outer lip	MNS	B	down	8.5
56	lower_t_lip_rm _o	Vertical displacement of midpoint between right corner and middle of top outer lip	MNS	B	down	8.6
57	raise_b_lip_lm_o	Vertical displacement of midpoint between left corner and middle of bottom outer lip	MNS	B	up	8.7
58	raise_b_lip_rm_o	Vertical displacement of midpoint between right corner and middle of bottom outer lip	MNS	B	up	8.8

FAP	Name	Description	Unit	Uni- / bidirectional	Motion	Feature point
59	raise_l_cornerlip_o	Vertical displacement of left outer lip corner	MNS	B	up	8.3
60	raise_r_cornerlip _o	Vertical displacement of right outer lip corner	MNS	B	up	8.4
61	stretch_l_nose	Horizontal displacement of left side of nose	ENS	B	left	9.1
62	stretch_r_nose	Horizontal displacement of right side of nose	ENS	B	right	9.2
63	raise_nose	Vertical displacement of nose tip	ENS	B	up	9.3
64	bend_nose	Horizontal displacement of nose tip	ENS	B	right	9.3
65	raise_l_ear	Vertical displacement of left ear	ENS	B	up	10.1
66	raise_r_ear	Vertical displacement of right ear	ENS	B	up	10.2
67	pull_l_ear	Horizontal displacement of left ear	ENS	B	left	10.3
68	pull_r_ear	Horizontal displacement of right ear	ENS	B	right	10.4

Table B.1: FAPs

Appendix C

FaceEditor parameter XML DTD

Under the root `faceeditor-parameters`, the four children `fp-positions`, `keep-syncs`, `deformers` and `vertex-masks` must occur once and in order. `fp-positions` contains one or more `fp-position` elements each of which having attributes to denote the feature point and the three coordinates. `keep-syncs` contains a plain space delimited list of FAP numbers for which synchronization must be kept. `deformers` contains elements that on their part contain the various parameters. For `ease-deformer`, all parameters are contained in attributes. `vertex-masks` contains one or more `vertex-mask` elements each of which having an attribute with the FAP number and containing a list of indices of vertices. See the DTD below.

```
<!DOCTYPE FEPARAM [

<!ELEMENT faceeditor-parameters (fp-positions, keep-syncs,
    deformers, vertex-masks)>
<!ELEMENT fp-positions (fp-position?)>
<!ELEMENT keep-syncs (#PCDATA)>
<!ELEMENT deformers (ease-deformer?, smooth-deformer?, deformer?)>
<!ELEMENT vertex-masks (vertex-mask?)>

<!ELEMENT fp-position>
<!ATTLIST fp-position x CDATA #REQUIRED>
<!ATTLIST fp-position y CDATA #REQUIRED>
<!ATTLIST fp-position z CDATA #REQUIRED>
<!ATTLIST fp-position fp CDATA #REQUIRED>

<!ELEMENT ease-deformer>
<!ATTLIST ease-deformer x CDATA #REQUIRED>
<!ATTLIST ease-deformer y CDATA #REQUIRED>
```

```
<!ATTLIST ease-deformer z CDATA #REQUIRED>
<!ATTLIST ease-deformer fap CDATA #REQUIRED>
<!ATTLIST ease-deformer size CDATA #REQUIRED>
<!ATTLIST ease-deformer scalex CDATA #REQUIRED>
<!ATTLIST ease-deformer scaley CDATA #REQUIRED>
<!ATTLIST ease-deformer scalez CDATA #REQUIRED>
<!ATTLIST ease-deformer use-vertex-mask CDATA #REQUIRED>
<!ATTLIST ease-deformer invert-vertex-mask CDATA #REQUIRED>
<!ATTLIST ease-deformer ease CDATA #REQUIRED>

<!ELEMENT smooth-deformer>
<!ATTLIST smooth-deformer x CDATA #REQUIRED>
<!ATTLIST smooth-deformer y CDATA #REQUIRED>
<!ATTLIST smooth-deformer z CDATA #REQUIRED>
<!ATTLIST smooth-deformer fap CDATA #REQUIRED>
<!ATTLIST smooth-deformer size CDATA #REQUIRED>
<!ATTLIST smooth-deformer scalex CDATA #REQUIRED>
<!ATTLIST smooth-deformer scaley CDATA #REQUIRED>
<!ATTLIST smooth-deformer scalez CDATA #REQUIRED>
<!ATTLIST smooth-deformer use-vertex-mask CDATA #REQUIRED>
<!ATTLIST smooth-deformer invert-vertex-mask CDATA #REQUIRED>
<!ATTLIST smooth-deformer smooth-center CDATA #REQUIRED>
<!ATTLIST smooth-deformer smooth-side CDATA #REQUIRED>

<!ELEMENT deformer>
<!ATTLIST deformer x CDATA #REQUIRED>
<!ATTLIST deformer y CDATA #REQUIRED>
<!ATTLIST deformer z CDATA #REQUIRED>
<!ATTLIST deformer fap CDATA #REQUIRED>

<!ELEMENT vertex-mask (#PCDATA)>
<!ATTLIST vertex-mask fap CDATA #REQUIRED>

]>
```

Bibliography

[1] Behavior markup language (bml) version 1.0 (draft). http://wiki.mindmakers.org/projects:bml:draft1.0.

[2] Online mpeg-4 fa implementation evaluation appendix. http://rcpaul.nl/ut/feeval/.

[3] *A muscle model for animation three-dimensional facial expression*, New York, NY, USA, 1987. ACM.

[4] G. A. Abrantes and F. Pereira. Mpeg-4 facial animation technology: survey, implementation, and results. *IEEE Transactions on Circuits and Systems for Video Technology*, 9(2):290–305, 1999. Cited By (since 1996): 35.

[5] Balci. Xface: Mpeg-4 based open source toolkit for 3d facial animation.

[6] Nikolaus Bee, Stefan Franke, and Elisabeth Andr. Relations between facial display, eye gaze and head tilt: Dominance perception variations of virtual agents. In *Proceedings of Affective Computing and Intelligent Interaction (ACII '09)*, 2009.

[7] Douglas DeCarlo, Corey Revilla, Matthew Stone, and Jennifer Venditti. Making discourse visible: Coding and animating conversational facial displays. In *In Proc. Computer Animation 2002*, pages 11–16, 2002.

[8] P. Ekman and W. Friesen. *Facial Action Coding System: A Technique for the Measurement of Facial Movement.* Consulting Psychologists Press, Palo Alto, 1978.

[9] P. Ekman, W. V. Friesen, and P. Ellsworth. Emotion in the human face: Guidelines for research and an integration of findings. page 191, 1972.

[10] N. Ersotelos and F. Dong. Building highly realistic facial modeling and animation: A survey. *Visual Computer*, 24(1):13–30, 2008.

[11] Irfan A. Essa, Massachusetts Institute of Technology. Media Laboratory. Vision, and Modeling Group. *Analysis, interpretation and synthesis of facial expressions*. PhD thesis, Vision and Modeling Group, Media Laboratory, Massachusetts Institute of Technology, Cambridge, Mass., 1995.

[12] Keith Waters Frederic I. Parke. *Computer Facial Animation*. A K Peters Ltd, Wellesley, MA, 1996.

[13] P. Kalra, A. Mangili, N.M. Thalmann, and D. Thalmann. 3d interactive free form deformations for facial expression. 1991.

[14] Nikita Kojekine, Vladimir Savchenko, Mikhail Senin, and Ichiro Hagiwara. Real-time 3d deformations by means of compactly supported radial basis functions. In *In Short papers proceedings of Eurographics*, pages 35–43, 2002.

[15] S. Kopp, B. Krenn, S. Marsella, A. N. Marshall, C. Pelachaud, H. Pirker, K. R. Thrisson, and H. Vilhjlmsson. *Towards a common framework for multimodal generation: The behavior markup language*, volume 4133 LNAI. 2006. Cited By (since 1996): 3.

[16] L. Malatesta, A. Raouzaiou, K. Karpouzis, and S. Kollias. Mpeg-4 facial expression synthesis. *Personal and Ubiquitous Computing*, 13(1):77–83, 2007.

[17] Carnegie Mellon University School of Computer Science. Facs - facial action coding system. `http://www.cs.cmu.edu/afs/cs/project/face/www/facs.htm`.

[18] Igor S. Pandzic and Robert Forchheimer, editors. *MPEG-4 Facial Animation: The Standard, Implementation and Applications*. John Wiley & Sons, Inc., New York, NY, USA, 2003.

[19] Frederic I. Parke. Techniques for facial animation. pages 229–241, 1991.

[20] Stefano Pasquariello and Catherine Pelachaud. Greta: A simple facial animation engine. In *In Proc. of the 6th Online World Conference on Soft Computing in Industrial Applications*, 2001.

[21] R. Plutchik. Emotion: A psychoevolutionary synthesis. *Emotion: A Psychoevolutionary Synthesis*, 1980. Cited By (since 1996): 347.

[22] A. Raouzaiou, N. Tsapatsoulis, K. Karpouzis, and S. Kollias. Parameterized facial expression synthesis based on mpeg-4. *Eurasip Journal on Applied Signal Processing*, 2002(10):1021–1038, 2002.

[23] Z Ruttkay. Constraint-based facial animation, 2001.

[24] M. Thiebaux, A. N. Marshall, S. Marsella, and M. Kallmann. Smartbody: Behavior realization for embodied conversational agents. *The 7th International Conference of Autonomous Agents and Multiagent Systems (AAMAS'08)*, 2008. Cited By (since 1996): 1.

[25] C. M. Whissel. The dictionary of affect in language. *Emotions, Theory, and Experience: The Measurement of Emotions*, 1989. Cited By (since 1996): 1.

[26] Y. Wu, P. Kalra, L. Moccozet, and N. Magnenat-Thalmann. Simulating wrinkles and skin aging. *The Visual Computer*, 15(4):183–198, July 1999.

[27] Y. Zhang, Q. Ji, Z. Zhu, and B. Yi. Dynamic facial expression analysis and synthesis with mpeg-4 facial animation parameters. *IEEE Transactions on Circuits and Systems for Video Technology*, 18(10):1383–1396, 2008.